

RL-TR-97-29
Final Technical Report
June 1997



DISTRIBUTED SYSTEM CONTROL

Lockheed Martin

**James A. Berea, Bharat Bhangava, Tom Geigel, Kane Kim,
and Jus-eak Kim**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

19970918 129

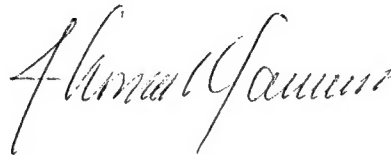
DTIC QUALITY INSPECTED 3

**Rome Laboratory
Air Force Materiel Command
Rome, New York**

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-29 has been reviewed and is approved for publication.

APPROVED:



THOMAS F. LAWRENCE
Project Engineer

FOR THE COMMANDER:



JOHN A. GRANIERO, Chief Scientist
Command, Control, & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/C3AB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE June 1997	3. REPORT TYPE AND DATES COVERED Final Jun 94 - Mar 96		
4. TITLE AND SUBTITLE DISTRIBUTED SYSTEM CONTROL		5. FUNDING NUMBERS C - F30602-94-C-0032 PE - 62702F PR - 5581 TA - 21 WU - AH		
6. AUTHOR(S) James A. Beres, Bharat Bhangava, Tom Geigel, Kane Kim, and Jus-eak Kim				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Lockheed Martin Advanced Technology Laboratories 1 Federal St., A&E, 3W Camden NJ 08102		8. PERFORMING ORGANIZATION REPORT NUMBER N/A		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory/C3AB 525 Brooks Road Rome NY 13441-4505		10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-97-29		
11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Thomas F. Lawrence/C3AB/(315) 330-2925				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) Global control in distributed systems had not been well researched. Control had only been addressed in a limited manner, such as for data-update consistency in distributed, redundant databases or for confidentiality controls (access control authentication). The purpose of control is to allocate the system's resources to the most important objective. The system must manage resources during system operation. Anomalies are conditions that obstruct the system from achieving user objective. Most research had been limited to a single-processor situation; however, research had to extend control to the distributed environment. The predictability of the external environment, communication delays, data accuracy (state information) and anomalies, and stability of decision algorithms constrain global control. The integration of control, which at various levels of granularity, would enable distributed-teleconferencing, and adapt to systems conditions, was investigated.				
14. SUBJECT TERMS Distributed Control, Quality of Service, Multimedia Distributed Systems, Resource Management		15. NUMBER OF PAGES 80		
		16. PRICE CODE		
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Table of Contents

Section	Page
1.0 Introduction	1
1.1 Program Rationale	1
1.2 Report Organization	1
1.3 Program Organization	1
1.4 Team Members	1
2.0 Overview of Distributed System Control	3
2.1 Program Objective	3
2.2 Design Philosophy	4
2.2.1 System Assumptions	5
2.2.2 Adaptation Types	5
2.2.3 Compliance of Distributed Systems Control	6
2.2.4 Value Functions	6
2.2.5 Global Management and Synchronization	7
2.2.6 Benefit-Loss Functions	7
2.3 Architecture of Distributed System Control	8
2.3.1 System Manager	9
2.3.2 Local Manager	10
2.3.3 tcp_comms	11
3.0 Video Teleconferencing Adaptability	13
3.1 Network Video Software	13
3.2 Adaptability Trades	13
3.3 Reconfiguration Policy	13
4.0 Implementation of the Demonstration	15
4.1 Hardware	15
4.2 Software	15
4.3 Demonstration Operation	15
5.0 Conclusions	17
6.0 Appendices	19
6.1 Adaptable Video Teleconferencing	21
6.2 Distributed System Control Algorithms	45
6.3 tcp_comms Interface and Users Guide	51
6.4 Demonstration User's Guide	53
6.5 Benefit Loss Scheduling	63

Table of Contents

List of Figures

Figure	Page
2-1 Distributed System Control's system architecture	3
2-2 Simple benefit-loss function	8
2-3 DSC system architecture	9
2-4 The SM's GUIs make system administration easy	10
2-5 The user monitors and controls applications with these GUIs	11
4-1 Demonstration architecture	16
6-1 Percentage of time encoding over total processing time for resizing	30
6-2 Percentage of time decoding over total processing time for resizing	30
6-3 Extended architecture of network video	32
6-4 Frame compression using pixel averaging to achieve 4:1 reduction	33
6-5 Frame rates for resized frames	36
6-6 Percentage of time encoding over total processing time for resizing	37
6-7 Percentage of time decoding over total processing time for resizing	37
6-8 Percentage of time encoding over total processing time for resolution reduction	38
6-9 Percentage of time decoding over total processing time for resolution reduction	38
6-10 Impact of momentum on DCT forward transform time	40
6-11 Impact of momentum on NV forward transform time	40
6-12 Impact of momentum on DCT reverse transform time	41
6-13 Impact of momentum on NV reverse transform time	41
6-14 Time spent in forward transform: NV versus DCT	42
6-15 Time spent in reverse transform: NV versus DCT	42
6-16 Compression ratio: NV versus DCT	42
6-17 Effect of recoding overhead on frame rates	44
6-18 Effect of changing recording frequencies on sizes of recordings	44
6-19 SM's LM window	55
6-20 SM's Start Application window	55
6-21 LM's GUI	55
6-22 LM's Control Panel window	56
6-23 LM's Application GUI window	57
6-24 Sample DSC Application resource file	59

Table of Contents
List of Tables

Table	Page
4-I System Configuration	15
6-I CPU use Under Various LoS	30
6-II Image Qualities for Different Compression Factors for Y and UV Components	35
6-III CPU Use in Various Codec Schemes Support by NV	39
6-IV Frame Rate Obtained Using Various Codec Schemes Supported by NV	39
6-V Generic Resource Values	47
6-VI Results of the Benefit-Loss Experiment	65
6-VII Classes of Kernel Threads	67

1.0 Introduction

This report documents work Lockheed Martin Advanced Technology Laboratories (ATL) performed July 1994 to May 1996 on the Distributed Systems Control (DSC) contract, F30602-94-C-0032. The U.S. Air Force's Rome Laboratory, Directorate of Command Control and Communications Computer Systems Technology Branch (C3AB), sponsored the contract.

1.1 Program Rationale

When the DSC effort began, global control in distributed systems had not been well researched. Control had only been addressed in a limited manner, such as for data-update consistency in distributed, redundant databases or for confidentiality controls (access control, authentication). The purpose of control is to allocate the system's resources to the most important objective. The system must manage resources during system operation. Anomalies are conditions that obstruct the system from achieving user objectives. Most research had been limited to a single-processor situation; however, research had to extend control to the distributed environment. The predictability of the external environment, communications delays, data accuracy (state information) and anomalies, and stability of decision algorithms constrain global control. The Advanced Technology Laboratories designed DSC to investigate the integration of controls, which at various levels of granularity, would enable distributed-systems control to operate, enable a multimedia application, enable video teleconferencing, and adapt to system conditions.

1.2 Report Organization

- Section 1 — Introduction, program objective, and team members
- Section 2 — Overview of Distributed System Control
- Section 3 — Adaptability of video teleconferencing
- Section 4 — Implementation of the demonstration
- Section 5 — Conclusions
- Section 6 — Appendices.

1.3 Program Organization

The DSC program is an outgrowth of research done on contract for Rome Laboratory's Adaptive Fault Resistant System (AFRS) and its predecessor, Adaptive Fault tolerance (AFT).

1.4 Team Members

Tom Lawrence, U.S. Air Force Rome Laboratory C3AB, directed the efforts of ATL, which was the program's prime contractor.

Other team members included Purdue University and the University of California, Irvine. Purdue University investigated new technology in video teleconferencing and made experimental modifications to Network Video (NV), a video teleconferencing application developed by Ron Frederick, Xerox PARC. The University of California, Irvine, investigated scheduling algorithms using benefit-loss functions.

2.0 Overview of Distributed System Control

This section describes Distributed System Control (DSC) and the rationale behind its development.

2.1 Program Objective

The objective of the program was to research and demonstrate integrated-control concepts for adaptability in distributed systems that reflected user expectations for a quality of service (QoS). Quality of service is timeliness, precision, and accuracy (TPA) of information provided to the user. Adaptability to anomalies in the system is inherent in the control software and the application. The work focused on adaptation and demonstration of video teleconferencing.

In general, anomalies in distributed systems include failure of a previously working component, hardware and software design failures, and shared-resource conflicts. Distributed System Control established a control architecture that enabled adaptation by system control and adaptation by distributed application. See Figure 2-1: The DSC architecture included a system manager, local host managers, system-monitoring software, user-value function data, application data, and a library of application-program interface calls.

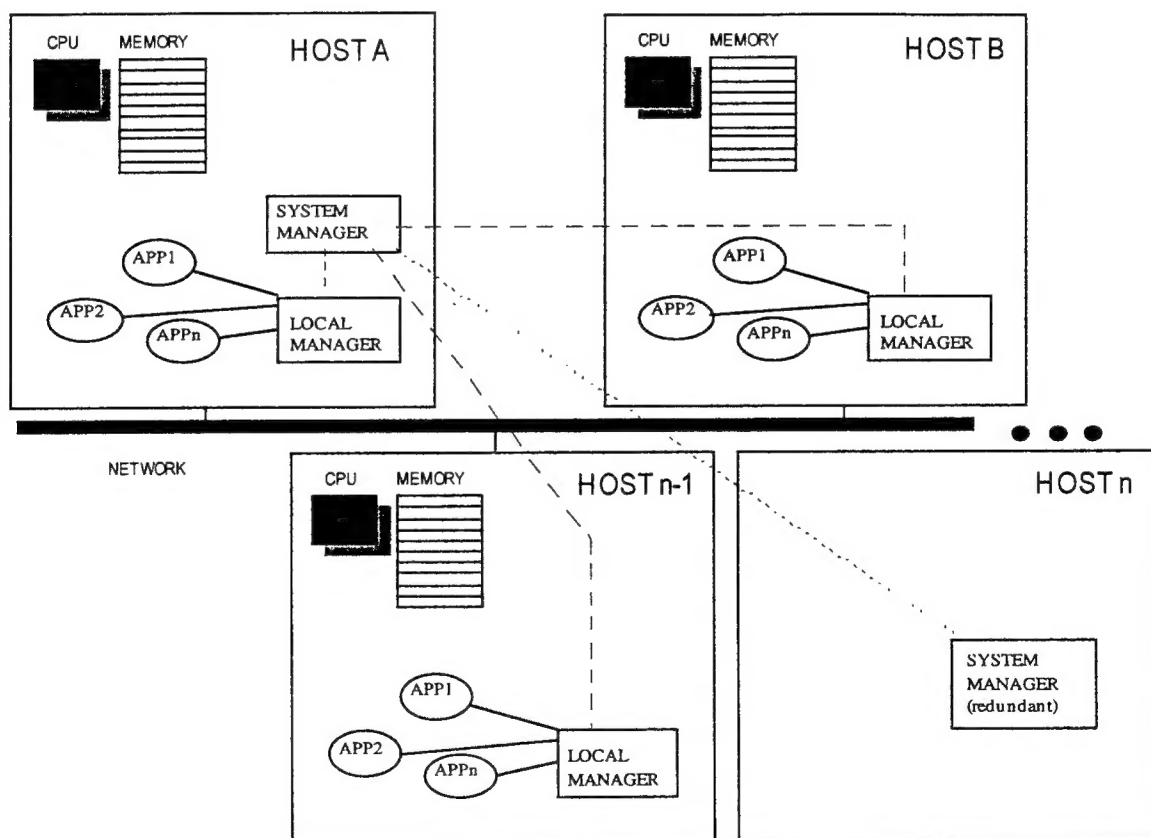


Figure 2-1. Distributed System Control's system architecture.

The DSC program developed and integrated control mechanisms that cooperatively controlled an application's dynamic requirements. The DSC characterized an

application's requirements by using objective functions made of value functions. The objective functions enabled the system's resources to collaboratively provide the best overall QoS. Using DSC, an application established distinct QoS operations called Levels of Service (LoS). The system initially established and maintained resource requirements for LoS during operation. The DSC controlled the QoS provided to the end user by controlling the application's LoS.

The DSC program investigated maximizing the overall value that the system provided to users. The Advanced Technology Laboratories (ATL) considered performance metrics (TPA) and the relationship of these metrics to the overall value a particular running application provided to the user.

The Advanced Technology Laboratories also considered the allocation of local and global resources to individual applications. Also investigated was the problem of maximizing the value returned by the services on a particular host. Finally, ATL suggested a hierarchical approach, building upon the individual host solution, in solving the global, system-wide problem.

2.2 Design Philosophy

The Advanced Technology Laboratories viewed distributed system control as a complex, realtime, optimization, and resource-allocation problem. It is realtime because anomalies of different kinds occur in realtime and user requirements may change dynamically. The set of resources remains relatively fixed, and it must be allocated to those user requirements that are most important at that time.

One possible approach is to have a complex, centralized resource manager that allocates based on input from all resources in the distributed system (system resources and application needs). This approach is the simplest to implement, but it is insufficient for most realtime decisions. The time-scale for different actions in the system covers a wide range; process scheduling may occur at the sub-millisecond range, while communications delays cannot be predicted to within tens of milliseconds. Having a centralized entity making communication-level or CPU-scheduling decisions is an unreasonable option.

An extreme approach would be a full peer-to-peer approach without centralized decision making. However, this is also impractical because it is too complex to agree with every peer when a decision has to be made.

There needed to be multiple levels of decision making, each with a well-defined set of services, timing granularity, and associated QoS. The ideal system may have schedulers for each resource that have the ability to interact with higher and lower level schedulers, as necessary.

The Advanced Technology Laboratories realized that most decisions within a node (processor) are local (principle of locality) and that the system must manage different time scales of interest. For actions where several nodes must interact, ATL used a centralized System Manager (SM). For allocating resources within a node, it used a Local Manager (LM) to perform most of the work. For very fine grained control (CPU), ATL investigated several algorithms based on benefit-loss functions (BLF).

2.2.1 System Assumptions

The Advanced Technology Laboratories developed DSC with a particular system model. The model best described future distributed systems that may benefit from DSC and it defined the scope of the problem that DSC tried to solve:

- **Very Large Complex Systems** — Present and future distributed systems contain hundreds, even thousands, of nodes; they will continue to increase as processor speed increases. These systems are configured as a wide-area network.
- **Diverse User Applications** — Each application varies in its importance relative to others in the system. These applications also have diverse criticality, availability, and realtime requirements.
- **Dynamic Operational Environment** — The DSC program was interested in networks that were prone to anomalies — frequent reconfigurations, drop-ins and drop-outs — due to the external environment in which they worked.
- **Limited Resources** — There is no way to evaluate *a priori* all possible execution scenarios and/or configurations.
- **Stochastic Behavior** — Assume that the applications are probabilistic to some degree.

Distributed control in existing systems is a difficult problem, considering the lack of available integrated-control mechanisms. Heterogeneous platforms further restrict integrated solutions due to differing operating-system "hooks and handles." The Advanced Technology Laboratories developed a philosophy to monitor, manage resources, and adapt applications to the distributed application. The philosophy centered on consolidating existing control mechanisms into a resource-management system that allowed local and global control.

2.2.2 Adaptation Types

The DSC recognized two types of system adaptation:

- **Type-I** — The system manages resources to completely meet an application's needs. Most developers today use and study type-I adaptation. Operating systems constantly schedule processes based on CPU use and input/output bandwidth. The system performs memory caching based on least-recently-used algorithms; this approach allows the more active processes a bigger share of local memory. The disadvantage is that an application will not run unless all of its resource needs are met. The system has only one set of application resources requirements, which are static and pre-calculated.

Providing sophisticated distributed-control mechanisms is only a part of solving the adaptability problem. As important is the application's ability to adapt to degrees of resource availability. Almost none of the applications developed to date take advantage of these control mechanisms; applications run in a certain way or not at all. Response time is the only degree of freedom.

The DSC program approach defined type-II adaptation as the ability of the application to run with different resource configurations. This may mean using different algorithms that require varying amounts of resource capabilities (CPU, communications, storage).

- **Type-II** — Applications adapt by changing their expectations. Type-II adaptation

requires the application to have pre-defined and multiple expectation levels; this results in different LoS associated with differing resource requirements. Different LoS allow more flexibility in maintaining an overall QoS among all distributed applications under DSC control.

During the project, ATL investigated both types of adaptation. The DSC program supported type-I by controlling the application's process-scheduling priorities. It supported type-II by using LoS and the ability of the application to reconfigure to different LoS in realtime.

2.2.3 Compliance of Distributed System Control

The concept of DSC compliance is only pertinent to applications that wish to be controlled by DSC. Compliant applications must be able to register, communicate, and be controlled by DSC. Implementing compliance requires modification to existing applications and it is only a small consideration for new applications. The DSC program defined an Application Programmer's Interface (API), described in Appendix 6.4.5, that provided controls to build a DSC-compliant application. The DSC program's benefits to the application relied on the reconfigurability and the number of LoS that the application could define.

The DSC's ultimate impact on providing QoS to distributed applications in an anomalous environment always increased when it controlled all distributed applications in that environment. However, ATL designed DSC to operate on a system with applications outside its control. As a resource-management system, DSC reacted to a non-controllable process that consumed resources as if they were unavailable due to a physical reason, such as a disabled network connection.

2.2.4 Value Functions

The original description for DSC introduced the concepts of TPA to describe the value of services provided by the system:

- **Timeliness** — Timing of required events
- **Precision** — Quantity of required data
- **Accuracy** — Compliance to semantics and contexts.

The Advanced Technology Laboratories identified the relationship of TPA to user value as a value function. By analyzing potential applications that may run under DSC, the range of possibilities for TPA often formed a set of discrete combinations, as opposed to a continuous function; for example: frame rate changed incrementally. Each configuration in which an application ran represented one possibility in this set. Because changing an application's configuration option could change associated levels of TPA, there was an inherent interdependence among them. Consequently, the approach ATL took to describe an application's value functions was the following:

- **Identified** each discrete combination of TPA values based on the various configurations in which an application could run
- **Assigned** a value to each configuration based on the relative importance that the configuration had to the user. This approach quantified LoS without the complexity of continuous functions.

2.2.5 Global Management and Distributed Application Synchronization

Resource management is a fairly simple task if the resources are local. However, decisions are complex when more than one host makes allocation decisions on global resources without regard to those made by other hosts. For example, two hosts could locally allocate 100 percent of the network's bandwidth to local processes, which results in a 100-percent over-allocation of the network. The global resource-allocation decision must take the other hosts' decisions into account.

The DSC program solved this problem by passing local allocation decisions to each host. For example:

Suppose the network is idle and Host A decides to run a network-intensive program. Consequently, the other hosts notice a decrease in network availability. If another more valuable process needs a currently unavailable portion of network bandwidth, then it will not be activated unless it is controlled by the same host as the first process. The host controlling the second process has no way of deciding that it has a better service to provide because it does not have information about the first process. The DSC program solved this problem by creating a virtual process within Host A's process table; Host A can now include that process in its resource-allocation calculations.

Another global-scheduling problem is when a distributed application's processes run on different hosts. A synchronization protocol must exist to guarantee that the application runs at the compatible LoS. To address this problem, the DSC program included the distributed process' LoS in the decisions that it made about local resource allocation. For example:

Similar to the global-management solution, each host passed information about its local process to the distributed application on all other hosts. Specifically, the system passed LoS that the local process ran between hosts. The hosts invalidated all other LoS for the distributed application's local processes except for the LoS that ran remotely. This synchronized the application by disallowing a process to run at a different LoS. As soon as a process' LoS could be upgraded, the other hosts received notification of the upgrade and made allocation decisions based on that LoS.

2.2.6 Benefit-Loss Functions

The Advanced Technology Laboratories investigated BLFs as a way to better control applications. To provide the expected QoS, some applications were configured into multiple threads of execution within the application. It was possible to manage multiple threads with native host schedulers by varying the threads' priorities. Each thread had an associated BLF, which determined how DSC's local scheduler prioritized the threads within its host.

Like value functions, BLFs determine how valuable a particular thread is to the overall system when it runs. Instead of relying on a thread's run value, the DSC accumulated a loss value if the thread did not run or if it missed its deadline. This function was

application-dependent and relied on a thorough understanding of the inner details of the application. Figure 2-2 is an example of a BLF. A static positive value resulted if a thread ran before its deadline; likewise, a static negative value resulted when it did not. By comparing various runtime-scheduling parameters (laxity, earliest deadline, or highest benefit-loss value), DSC controlled the application's thread at the kernel-scheduler level. It did this by tracking the accumulation of the BLF over time. If the BLF reached a user-defined threshold, then the application could prompt DSC to adapt the application to a different LoS in the same way the application would adapt to a resource constraint. Therefore, in addition to providing a fine-grain level of control for DSC, an application-controllable adaptation mechanism can be implemented.

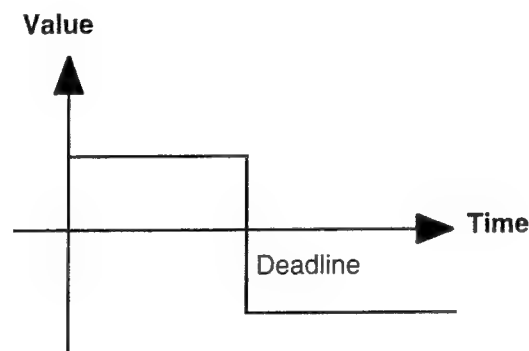


Figure 2-2. Simple benefit-loss function.

Appendix 6.5 discusses two items: implementation of a BLF scheduler that ATL investigated during the contract and results of tests using different implementations. The Advanced Technology Laboratories compared three schedulers: one based on least laxity (time remaining before the event deadline), one on highest benefit-loss value, and one on highest ratio of benefit-loss value to laxity. The comparison showed that the ratio of benefit-loss value to laxity was superior. The Advanced Technology Laboratories did not implement BLFs in DSC because of a threading bug it discovered in version 5.4 of Sun's Solaris operating system. Appendix 6.5 also reports results of tests ATL performed on version 5.3.

2.3 Architecture of Distributed System Control

The DSC system is hierarchical, so it supports different levels of control (Figure 2-3). At the top level, there is an SM, whose primary task is to monitor and control distributed DSC applications and system resources at a global level. One of DSC's goals was to maintain distributed application-execution in a faulty environment; therefore, ATL designed the SM to be redundant.

An LM resides on every host. Its primary task is to control and monitor DSC applications and resources at a local level. The LM maximizes the use of resources based on the value of the application. It does this by allocating and scheduling the resources of DSC's processes. The LM also executes reconfiguration and start/stop commands it receives from the SM.

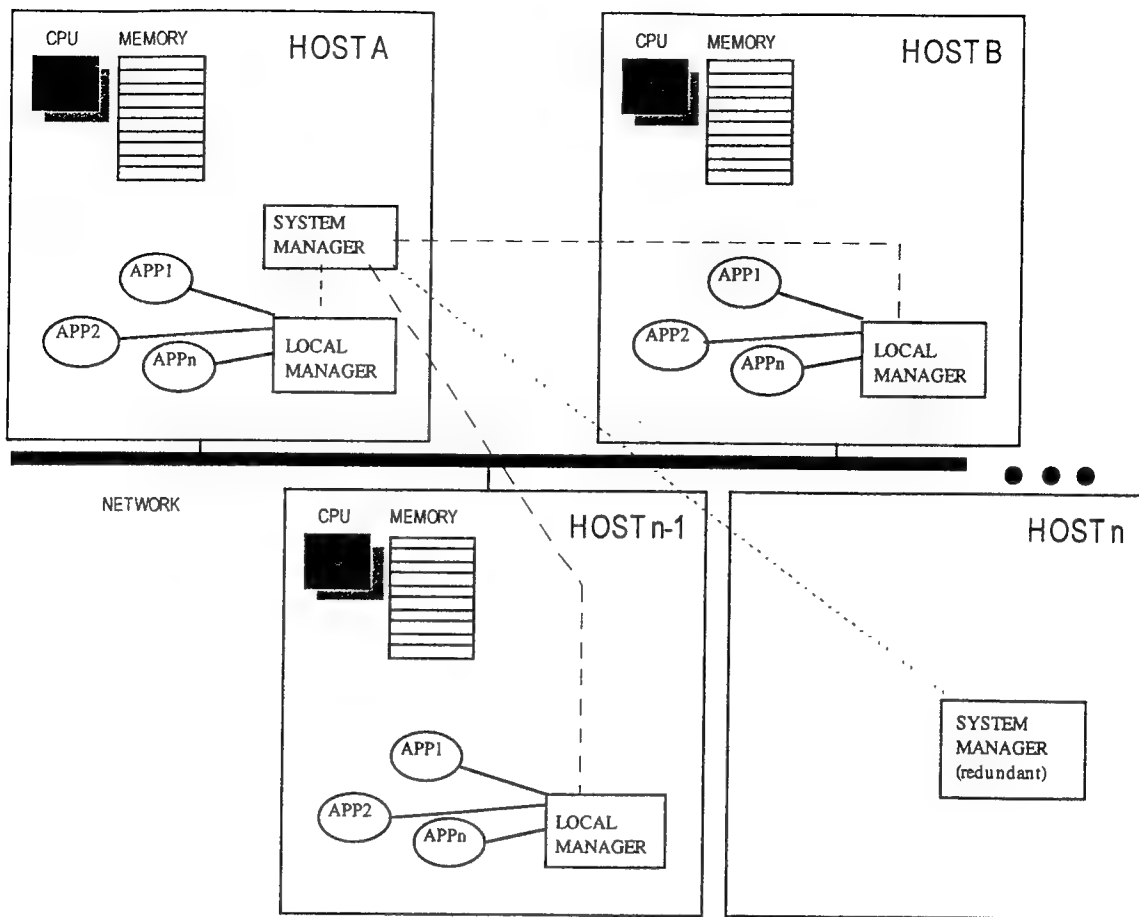


Figure 2-3 DSC system architecture.

For the initial development and implementation of DSC, the system managed CPU, main memory use (local resources), and network bandwidth (a global resource).

The DSC software components and DSC-controllable applications communicated via UNIX sockets using the `tcp_comms` communication package. The Advanced Technology Laboratories developed `tcp_comms` to provide a fast, multicast-process-group communication capability using transmission control protocol (TCP)/Internet protocol (TCP/IP).

2.3.1 System Manager

The SM controls DSC applications at the global level. Its responsibilities include the following:

- **Monitoring** system-level (global) resources
- **Monitoring** and handling system-level connectivity between hosts
- **Coordinating** start/stop of DSC applications
- **Providing** a graphical user interface (GUI) for system administration.

The initial DSC design had the SM manage global policy. Application performance and local-resource-use feedback from the LMs would be analyzed by dynamically-

changing global-resource management policies. The LM would determine whether resources were being assigned to the most important distributed DSC applications. The SM commanded the LMs to reconfigure the DSC application processes, thereby dictating global resource-management policy. However, the timeliness of sending and synchronizing reconfiguration commands among hosts limited the usefulness of SM-based centralized control for realtime applications, such as video teleconferencing. The complexity involved in implementing any process-migration capability, a necessary coarse-grained reconfiguration tool, was beyond the program. For these reasons, ATL implemented a decentralized LM-based approach to global-resource management. Nonetheless, the SM was still the central location for starting and stopping DSC applications, as well as the location for monitoring network connectivity.

See Figure 2-4: The SM's GUI provided DSC's basic global-level control functions. It monitored the availability of each LM on each host, and it allowed the user to control the application from a central location.

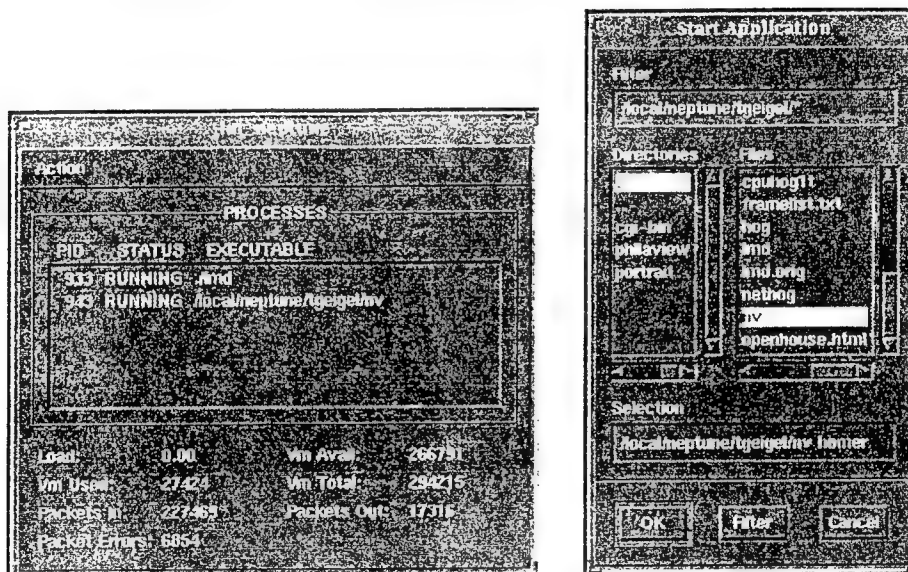


Figure 2-4. The SM's GUIs make system administration easy.

Appendix 6.4.2 details the SM's GUI.

2.3.2 Local Manager

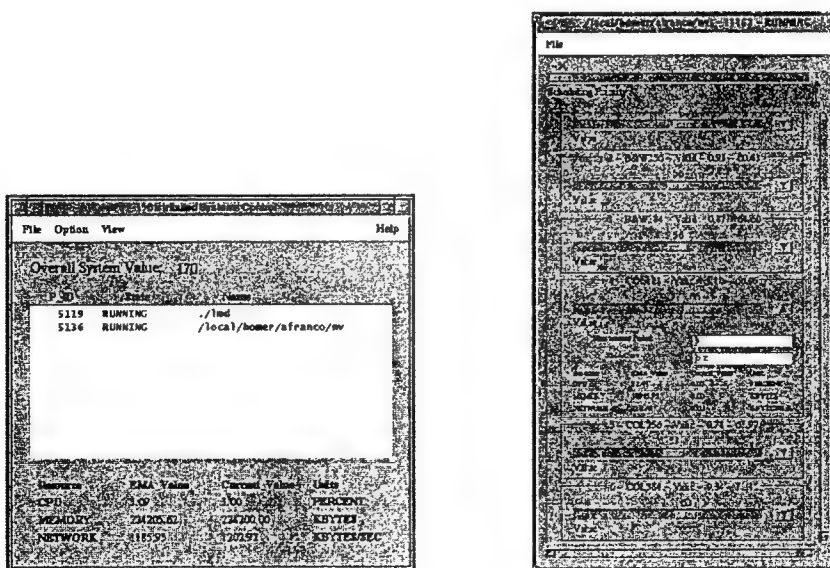
The LM is the parent process for all DSC-controlled processes. Its primary function was to allocate computing resources to processes. The reason for the allocation was to maximize the aggregate value of DSC applications running on the host without overusing available resources. The LM also coordinated its decisions with other hosts' LMs to ensure that distributed DSC applications ran in a consistent configuration. In DSC, LM allocated CPU use, core memory, and network bandwidth. However, there were no theoretical restrictions on including other shared computer resources, such as video cards or communication ports. The LM responded to systematic (e.g., process aborted) and application-specific application errors (e.g., an application reported to DSC to reconfigure when an application state changed).

The LM made allocation decisions on a cyclic basis (measurement period) based on user-supplied descriptions of DSC-controlled applications. A typical DSC application was reconfigurable into a number of LoS. (Refer to Figure 6-24, which describes each LoS.) The table contained elements describing the value of the applications' configuration and its estimated resource requirements for CPU use, memory, and network bandwidth. For every measurement period, the LM calculated a value/cost ratio for each LoS of each runnable DSC application. It based the calculation on the LoS's value and the predicted resource use for the application. It calculated predicted resource use by using an exponential, moving, average filter that accounted for past history and present use. The LM sorted DSC's applications on value/cost ratio and selected the highest LoS for each application that still ran with the remaining resources.

Appendix 6.2 details the above methods.

See Figure 2-5: The LM has a GUI that allowed developers and administrators to control various parameters of LM and other applications. The LM GUI gave a list of active processes it controlled and current measurements for CPU, memory, and network resources. By double-clicking on any application, an application's LoS GUI appeared. The GUI allowed the user to adjust the value of each defined LoS, measurement period, and EMA_constant, which directly weighted previous measurements in present resource calculations.

Appendix 6.4.3 details the LM's GUI.



Figures 2-5. The user monitors and controls applications with these GUIs.

2.3.3 tcp_comms

The Advanced Technology Laboratories developed `tcp_comms` as a small, fast, UNIX, socket-based communications package. All DSC software components and DSC-controllable applications communicate using `tcp_comms`. `tcp_comms` provides a communication substrate that allows multicast TCP/IP communication using a process

group communication model. `tcp_comms` abstracts message passing to a level where the software developer only has to create the message and send it to the correct process group. `tcp_comms` maintains group names and members via the nameserver (NS) daemon.

Basic execution of `tcp_comms` process occurs in the following sequential manner:

- The application must use `tcp_comms_init()`, which creates a point-to-point TCP socket from the process to the NS.
- The application joins communication groups via `tcp_comms_join_group()`.
- `tcp_comms_join_group()` sends a message to the NS, which registers the application as a member of that group and returns a list of other processes that are also group members.
- `tcp_comms_join_group()` then creates point-to-point TCP sockets for those processes, if they have not already been established. If one of these processes aborts, then the application and the NS will be notified (via the death of the TCP port), and the group member information will be updated automatically by the NS.
- Finally, `tcp_comms_msg_send()` sends a byte-array buffer to any group. `tcp_comms_wait_for_next_msg()` and `tcp_comms_get_next_msg()` provide blocking and non-blocking ways of receiving messages from a group.

If one wishes to send messages to a particular group but not become a member of that group, `tcp_comms_get_group_id()` can be used.

`tcp_comms` is thread-safe and assumes that the user will use threads to multiplex reception of messages from different groups.

Appendix 6.3 details `tcp_comms` API.

3.0 Video Teleconferencing Adaptability

3.1 Network Video Software

The Advanced Technology Laboratories (ATL) developed Distributed System Control (DSC) to provide distributed application adaptability. The Advanced Technology Laboratories demonstrated this concept with a video teleconferencing application. The DSC used network video (NV) — a popular video-conferencing tool developed at XEROX PARC. The Advanced Technology Laboratories chose NV because of its availability and functionality: It transmits and receives video data across the Internet and it runs over a wide range of network bandwidths. The NV uses a video-compression scheme that performs at a reasonable speed when implemented in software. It takes advantage of similarities between consecutive frames in a typical video stream and neighboring pixels in any region of the frame. The NV periodically transmits stationary blocks to improve image quality. The NV has three main modules: grabber, which grabs frames using a platform-specific video-capture card; encoder, which does lossy compression; and sender, which transmits data.

3.2 Adaptability Trades

The NV adapts color depth, frame resolution, frame size, and code scheme. For DSC, ATL used combinations of these adaptability features to establish levels of service (LoS), such as greyscale with 128-bit resolution and small frame size; or color with 256-bit resolution and medium frame size. The Advanced Technology Laboratories augmented the NV with three modules: network probe, admission control, and VC adaptability collaborator. It also ran performance trades to establish resource-use (CPU and communications) parameters for differing compression approaches. The objective was to establish unique LoS with unique resource requirements. The analysis also yielded unexpected results: In some cases, system developers perceived the clarity of the video produced with greater compression as better than that produced with less compression.

Section 6.1 details the NV studies.

3.3 Reconfiguration Policy

Based on adaptability trades, ATL developed policy guidelines for reconfiguring video-conferencing to meet timeliness, precision, and accuracy (TPA) goals when a system operated under resource constraints. These guidelines were the following:

- **Maintain Timeliness** when bandwidth decreases:
 - Reduce frame size
 - Reduce frame resolution
 - Dither color frame to black and white
 - Compress color depth
 - Switch to a code scheme that has a higher compression ratio.
- **Maintain accuracy** when bandwidth decreases:
 - Switch to a lossless code scheme with reduced frame size
 - Dither color frame to black and white
 - Compress color depth
 - Do not use lossy code schemes
 - Do not reduce frame size or resolution by a large factor
- **Maintain timeliness** when CPU-use increases:
 - Switch to a code scheme that requires less computation

- Reduce frame size
- Dither color frame to black and white
- Do not compress color depth
- Do not reduce frame resolution
- **Maintain accuracy** when CPU use increases:
 - Switch to a lossless code scheme
 - Reduce frame size
 - Dither color frame to black and white
 - Do not compress color depth
 - Do not reduce frame resolution
 - Do not use lossy code schemes.

Section 6.1 details the NV studies.

4.0 Implementation of the Demonstration

4.1 Hardware

The Advanced Technology Laboratories (ATL) demonstrated the Distributed System Control (DSC) on a network of three Sun computers connected by Ethernet™. The network video (NV) video-teleconferencing tool required video cameras and interface cards. The Advanced Technology Laboratories configured Sun's SunVideo card and camera on two machines for this purpose. The machines with the SunVideo cards required at least 64MB memory. The third machine required the power of a Sun SparcStation 10 or higher with 64MB memory; the Advanced Technology Laboratories used a Sun SparcStation 20 due to its availability. Table 4-1 lists the configuration of the Sun SparcStations.

Table 4-1. System Configuration

Machine	Type	Monitor	Disk	Memory	SunVideo
Neptune	Sparc10	20" Color	525MB	64 MB	Yes
Homer	Sparc 10	20" Color	525MB	98 MB	Yes
Denali	Sparc 20	20" Color	1.2GB	64 MB	No

4.2 Software

The necessary software consisted primarily of Sun Software Development packages. The Advanced Technology Laboratories installed SunVideo software on the machines that contained the video cards. The software is standard on Sun's Solaris operating system, version 5.4, but it is not a standard install option (Ref. *SunVideo Users Manual* for more details). The NV and DSC software are deliverables. The following was the required Sun software:

- Solaris operating system, version 5.4
- SparcWorks' C Compiler and Software Developer's Kit, version 3.0.1 or better
- OpenWindows, version 3.0 or better
- Motif 1.2 (SUNWmotif package comes with Sun OpenWindows)
- SunVideo Software package (SUNWrtvc SUNWrtvcu packages).

4.3 Demonstration Operation

Figure 4-1 shows the demonstration architecture. The primary application NV ran on two machines, each with a SunVideo card and camera. A third machine (Denali in the figure) ran an X Windows server; it provided a central location for viewing and controlling the dynamic reconfiguration of NV on camera-supplied machines as the user added worker applications. Worker applications were simple resource consumers that varied available resources: *Nethog* used network resources and *cpuhog* consumed CPU bandwidth.

The demonstration showed how DSC adapted the NV application based on available resources. As each worker application started, the system reconfigured NV to run at a different LoS — from full-frame-rate color to half-frame-rate black and white video.

Appendix 6.4 explains how to boot DSC software components and how to run and control DSC-compliant applications for the demonstration.

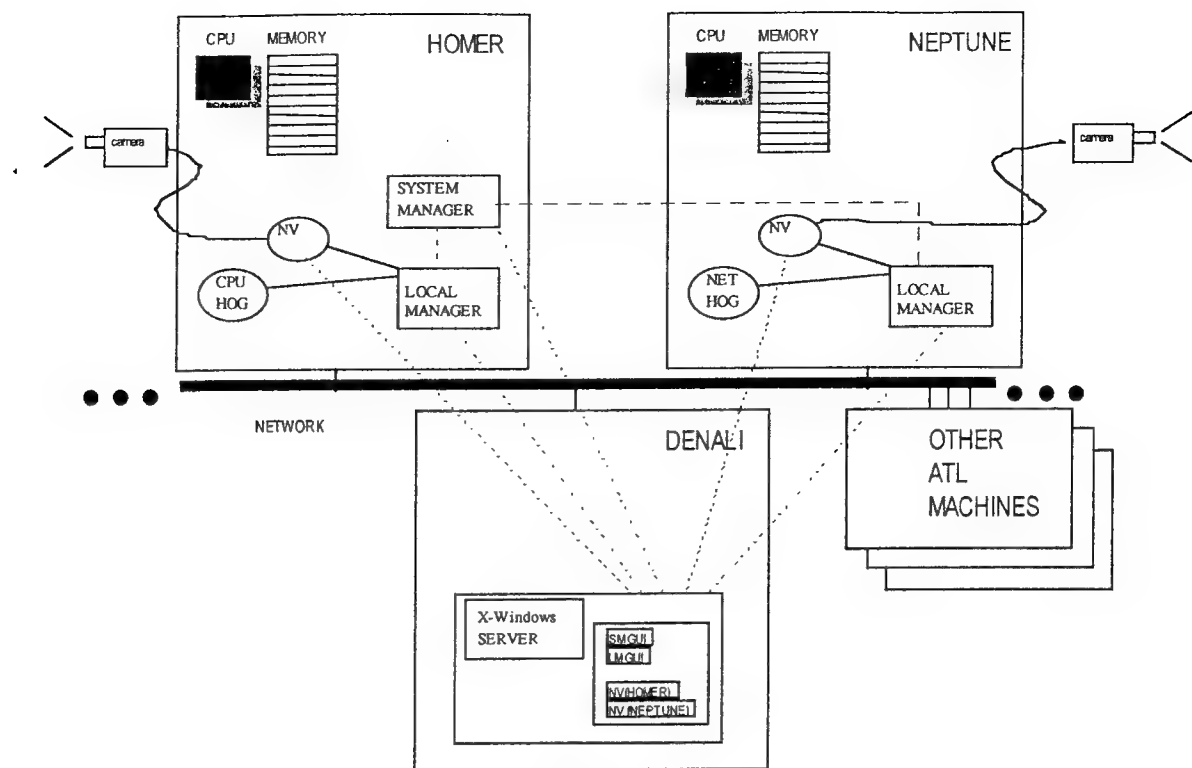


Figure 4-1. Demonstration architecture.

5.0 Conclusions

The Distributed Systems Control (DSC) program addressed the establishment of a set of controls that adjusted the quality of service (QoS) provided to the user based on the resources available to the system. The work focused on demonstrating controls with a multimedia application. The Advanced Technology Laboratories (ATL) established controls in a hierarchical structure to enable scalability, and it kept them simple to let the user understand and use them and to minimize system impact from intrusive monitoring.

The DSC used two types of adaptability:

- **Type I** — The application did not change its operation; rather the system adapted the resources to provide those needed by system requirements.
- **Type II** — It complements type I. The application adapted to operate to a different QoS based on considerations of timeliness, precision, and accuracy (TPA) embodied in the user's assessment of value. The Advanced Technology Laboratories based the fundamental concept of effective system adaptability of DSC on predictable performance. The DSC took advantage of NV's stability to demonstrate cooperative adaptability between the system and the application — termed type-II adaptability.

The Advanced Technology Laboratories could not directly control the application's QoS through different TPA parameters at runtime. Instead, it selected and calibrated levels of service (LoS) to implement QoS — discrete application configurations — for anticipated resource needs. Once ATL mapped QoS into an LoS and its associated system resource needs, the DSC managed system resources according to user preferences.

The Advanced Technology Laboratories selected unique LoS operating points for the video application: They were unique in the service provided and the resources required. The distinctions in service provided choices to the user for service and the distinct resource requirements provided meaningful choices to the control algorithms.

Significant to the approach of building discrete LoS was the need for extended design-time evaluation of QoS; that is, what services were meaningful to the user and how could they be characterized in terms value? Analysis is needed to clarify users' adaptability requirements in the same way other requirements are examined. For video-conferencing, resource requirements were relatively consistent for each discrete LoS, so DSC's adaptation algorithms took advantage of that predictability.

Instability in communication delays limited the ability to use fine controls. However, asynchronous-transfer-mode (ATM) communications provided the capability to specify QoS; therefore, it was possible to couple an application's QoS (as specified by an LoS) with a communication's QoS. This technique could be investigated as a way to implement various combinations of coarse and fine controls.

The Advanced Technology Laboratories investigated the feasibility of fine-grain scheduling based on benefit-loss functions. It found that a scheduling algorithm that used a ratio of the value-to-laxity scheme was superior in reducing the cumulative

benefit-loss over time. However, two steps would need to be taken to use such fine-grain controls:

- **Scheduling algorithms** would have to be implemented in the kernel to operate efficiently.
- **Detailed analysis** of the application would be required to break it into defined blocks with known timing requirements.

In summary, the DSC program demonstrated the ability to control system performance of video-conferencing. It did this by using a cooperative control methodology that had system and application adaptability. The DSC program highlighted the need for detailed adaptability design for type-II adaptability and defined conditions that would enable further integration of controls.

6.0 Appendices

Appendices in this report are the following:

- 6.1 Adaptable video teleconferencing
- 6.2 Distributed System Control algorithms
- 6.3 `tcp_comms` application programmer's interface and user's guide
- 6.4 Demonstration user's guide
- 6.5 Benefit-Loss scheduling

6.1 Adaptable Video Teleconferencing

6.1.1 Introduction

Video conferencing systems are practical in commercial and research institutions because the technology has advanced in networking and multimedia applications. A video conferencing session involves multiple parties who might be geographically dispersed, and who exchange realtime video data. However, anomalies such as site failure and network partitioning determine the effectiveness of the communication capabilities. Video conferencing systems cannot dynamically adapt to variations in system resources, such as network bandwidths, CPU use, and memory and disk storage. For video conferencing systems, only users can change parameters, such as frame sizes, codec schemes, color depths, and frame resolutions. These changes cannot be made automatically based on system measurements of currently available resources. It is necessary to limit the user's burden in keeping the system running in the most suitable mode and to provide the best possible service based on the status of the system. Incorporating adaptability into video conferencing systems minimizes the effect of variations in system environments on the quality of video-conference sessions.

The paragraphs in this section discuss the following:

- **Section 6.1.2** describes the concept of adaptability and how to achieve it in a video-conferencing system.
- **Section 6.1.3** describes common anomalies encountered in a distributed system.
- **Section 6.1.4** overviews the Network Video (NV) conferencing-system testbed, then describes extensions and modifications to NV and some reconfiguration issues.
- **Section 6.1.5** summarizes experimental data analyses, observations, and discussions.

6.1.2 Adaptability

Adaptability and reconfigurability are necessary to deal with the performance and reliability requirements of a system. Current distributed systems provide a rigid choice of algorithms to implement application software. Users make design decisions based on criteria, such as computational complexity, simulations under limited assumptions, and empirical evidence. The desired life-cycle of a system is at least several years. During such time, new applications surface and the technology advances, making earlier design choices less valid. Also, during a 24-hour period, users encounter a variety of load mixes, response-time requirements, and anomalies. An adaptable distributed system can meet the various application needs in the short term, and it can take advantage of advances in technology over the long term. Such a system will adapt to its environment during execution and it will be reconfigurable for new applications.

Adaptability is particularly important for mission-critical and distributed systems. Mission-critical systems push current hardware and software technology to their limits to meet extreme fault-tolerance or realtime performance requirements. Distributed systems must be adaptable to support heterogeneous hosts and to provide graceful degradation during anomalies. Computer networks support a larger and more diverse user community than centralized systems. Hosts of many different types running

different software will be required to support this community. Therefore, the underlying distributed system must be adaptable to a wide range of hardware and software.

Furthermore, failures that would require complete shutdown of a centralized system can often be tolerated with only slight performance degradation in a distributed system. The distributed-system software must be designed to be adaptable to a wide range of modes of operation, corresponding to the wide range of possible failure modes. For instance, a paper on dynamic quorum assignments¹ describes an algorithm for responding to failures by dynamically adjusting quorum assignments. As a failure continues, the system modifies more and more quorum assignments. When the failure is repaired, quorums that were changed can be brought back to their original assignments. By dynamically adapting to the failure, the availability of data in the system is increased at a cost that is only incurred during failure or recovery.

The principal advantages of adaptability are in reliability, performance, and software enhancement and maintenance. Adaptability provides for reliability through a design that is tolerant of site and communication failures. Adaptability improves performance because the system can adjust its transaction-processing algorithms for optimum processing of the current mix of transactions. Adaptability simplifies software enhancement and maintenance through a design that is oriented from the start to incorporate new ideas and approaches.

Subsystems can be replaced without affecting other parts of the system, and the design of each subsystem supports implementation of new algorithms. With maintenance costs climbing as high as 80 percent of life-cycle costs for many systems, a design that supports future changes is becoming an essential part of software development.

6.1.2.1 Flavors of Adaptability

There are four broad categories of adaptability: structural static, structural dynamic, algorithmic, and heterogeneous:

- **Structural-static** adaptability consists of software-engineering techniques to develop system software that can adapt to requirements changes over its life-cycle. Structural-static techniques are categorized differently if they refer to layered software or unlayered software:
 - **Layered software** supports vertical adaptability, which is the ability to replace a layer without affecting the other layers. Some layered software also supports horizontal adaptability, which is the ability to replace components within a layer.
 - **Unlayered software** can also take advantage of structural static adaptability. Fault-tolerance adaptability in hardware is classified into circuit-level and module level. Circuit level adaptability corresponds to software techniques that check for errors within a software module, using redundancy or correctness criteria. Module-level adaptability corresponds to techniques that isolate errors to within a single software module and allow the rest of the system to continue processing despite partial failures. Failure modes at the module level are often assumed to be fail-stop, which means that either the

¹ Bharat Bhargava and Shirley Browne. Adaptability to Failures Using Dynamic Quorum Assignments, Technical Report CSD-TR886, Purdue University, June 1989.

module does not produce a result or it produces the correct result. Module- and circuit-level adaptability in software also refer to the implementation techniques that support replacement of software modules or algorithms, respectively.

- **Structural-dynamic** adaptability, usually called reconfiguration, is restructuring a running system in response to failures or performance requirements. Reconfiguration includes site-failure protocols, network-partitioning protocols, and other techniques to reorganize the way the sites in a distributed system work together. For instance, performance reconfiguration includes dynamically changing the communications structure of processors in a non-shared-memory multiprocessor in response to changing tasks. Reconfiguration also has long-term benefits. Users can easily integrate new hardware into a running system, and port the software to new architectures with different interconnections between processors.
- **Algorithmic** adaptability is a set of techniques to dynamically change from the execution of one algorithm for a module to a different algorithm. For instance, a transaction system can change to a new concurrency controller, or a distributed system can change to a new site-failure algorithm. Algorithmic adaptability can take place in one of three ways:
 - **Temporal** adaptability refers to changes in algorithms over time. This method generally has a brief conversion period after which operation continues with the new algorithm.
 - **Per-transaction** adaptability consists of methods that allow each transaction to choose its own algorithm. Different transactions running at the same time may run different algorithms based on their requirements.
 - **Spatial** adaptability is a variant of per-transaction adaptability in which transactions choose the algorithm based on properties of the data items they access. Spatial adaptability is an advantage in cases in which properties of different algorithms are desired for different data items.
- **Heterogeneity** deals with the issues involved in distributed computing using many different types of computing systems. The simpler problems of heterogeneity include establishing physical connection between the machines and resolving data-type differences (e.g., size of integers, byte order). More difficult problems include getting diverse database systems to work together and developing software that takes advantage of the particular strengths of each machine. Heterogeneity is an important problem for the future, because missions requiring several different types of computing engine will become more common. Systems that can incorporate heterogeneous hardware and software components will have the advantage of being able to incorporate new technologies more easily. Solving the problem of heterogeneity involves many of the problems in algorithmic and reconfiguration adaptability; it is aided over the long term by the software-design techniques of structural static adaptability.

6.1.2.2 Techniques for Adaptability

Adaptable software can be supported by an infrastructure of software tools. Support for modular design can make it easier to implement adaptability. The principle behind modular design is that abstract entities in the software design should be represented as physical entities in the implementation. For instance, separate activities should be represented as separate processes. In many operating systems, such as UNIX, this

means that the activities are represented as separate address spaces, each with a single thread of control. Shared resources are placed in one of the address spaces and accessed from the other address spaces via messages or remote-procedure calls. Because messages and remote-procedure calls usually cost an order of magnitude more than local accesses, the performance of the entire system suffers. The problem is that the operating systems only offer a single-process abstraction. More flexible process abstractions make sharing of resources, such as memory and files, efficient and easy. For instance, some operating systems support multiple threads of control within a single address space.

Powerful remote communication primitives can also support adaptability. For instance, efficient multicast encourages the development of applications that can be spread among hosts in many different ways. Using logical multicast addresses, the application does not have to worry about the location of the destination. Servers can relocate without informing their clients. Any of a group of replicated servers can answer service requests directed to the logical address.

New languages also support adaptability. Object-oriented languages provide data abstraction to clearly define the interface to the programmer. The syntax and semantics of object references are the same whether the message is to an object in the same address space, in a different address space on the same computer, or even on a different computer. A smart compiler can exploit this flexibility by clustering objects into processes according to reference patterns, so most object invocations can be simple procedure calls. In fact, the object can be dynamically moved during program execution as long as the interface is not affected.

Models of adaptability are another tool. A sequencer model formalizes three basic approaches to algorithmic adaptability: generic state, converting state, and suffix-sufficient state. The model reduces the problem of adaptability to mapping the subsystem to a sequencer, choosing one of the adaptability approaches, and implementing it. Software engineers have studied design techniques that enhance structural static. Models for other types of adaptability are needed.

6.1.2.3 Using Adaptability

Incorporating adaptability into a system requires an understanding of how and when adaptability will be used. Structural-static adaptability is used every time the software is changed. Any software that will include maintenance in its life-cycle should incorporate techniques to make maintenance easier to reduce its life-cycle cost. Reconfiguration adaptability is necessary in systems for which availability is important or reconfiguration is common to maximize availability. Reconfiguration adaptability may also be necessary for mission-critical systems that must use different configurations to respond to different situations to perform correctly. Algorithmic adaptability is most important in systems that are pushing hardware performance to the limit or that require very high levels of fault tolerance. Algorithmic adaptability is useful for responding to environmental changes that are predictable or that last long enough to amortize the cost of the adaptation. Finally, support for heterogeneity is important in any system that has a long life-cycle or that will run on a large network of computers to maintain flexibility. In choosing when adaptability can be applied, there is

a trade-off between the costs and benefits. Experimental work is needed to evaluate adaptability in practical environments.

6.1.2.4 Adaptability in Video-Conferencing System

Adaptability is a very important concept in distributed systems. It means that a distributed system should be able to reconfigure itself. The reconfiguration is necessary to adjust to changes and anomalies in the system environments so it can provide services whose level is closest to the level of service (LoS) specified by the applications. Because there are various kinds of anomalies in a distributed system that can affect the services provided by the system, several levels of adaptability are needed to achieve this goal. In particular, a video-conferencing system should provide some policies and mechanisms to make it adaptable to the anomalies based on the available resources. The advantages of the adaptability schemes for video-conferencing systems include:

- **Heterogeneity** — A video-conferencing system that will adapt to heterogeneous environments; that is, a video-conferencing session can be held on different hardware platforms and different networks.
- **Scalability** — A video-conferencing system that will adapt itself as more users and more sites join a video conference in progress.
- **Anomaly management** — A video-conferencing system that will adapt to anomalies and degrade gracefully when available resources decrease or become unavailable.
- **Resource management** — A video-conferencing system that can make efficient use of resources, such as storage, CPU time, and communication bandwidth.

The basic idea for achieving adaptability for video-conferencing systems is to trade-off some aspects of video quality for others. For example, the frame rate decreases as the available network bandwidth drops. Because the smoothness of a video session is sometimes more important than any other aspects of video quality, users may have to maintain a reasonable frame rate during a video-conference session, even though the network performance degrades. To achieve this, users must sacrifice some aspects of video quality, such as color or resolution of video frames.

Adaptability can be achieved either by user intervention or by the system itself. If user intervention is required, then the system accepts inputs from the user and changes the LoS or some system parameters according to the new specification. If adaptability is automatically done by the system, then the distributed-control system periodically measures the available resources and supplies these parameters to the video-conferencing system. Based upon the current parameters, the video-conferencing system reconfigures itself in a user-transparent way to provide the best possible service based on some user-specified criteria that must be satisfied.

6.1.3 Anomalies in a Distributed System

Video-conferencing tools are inherently used on top of distributed systems. The anomalies in distributed systems will reduce the effectiveness and use of a video-conferencing tool. Typical distributed-system anomalies include the following:

- **System Resource Limits** — System resources are usually limited and change

dynamically. Common limited resources include available network bandwidth, CPU time, and disk storages:

- **Network bandwidth** in a distributed system is available in limited quantities and varies with network traffic. For example, the bandwidth of a typical Ethernet™ is 10 Mbps. Available network bandwidth may decrease drastically during a short period of time when many users simultaneously send large objects (packets). This is typical for video-conferencing sessions. When the available bandwidth drops sharply, the video-conferencing system can change the LoS by changing color or frame resolution to reduce the object's size.
- **CPU time** available for the video-conferencing processes changes depending on the system load. The system may choose to avoid compressing the data before sending to decrease the CPU use if the available bandwidth is large enough.
- **Disk storage** availability varies from machine to machine in distributed system. For example, a server in an organization may have several gigabytes of disk space and over 48 megabytes of memory, while a personal computer participating as a client may have only 500 megabytes of disk space and 4 megabytes of memory. In a heterogeneous environment like this, the system may choose to provide different kinds of service for each type of machine by negotiating before connecting. For example, the server may decide to reduce the frame rate or frame resolution if the participating site is a personal computer, while maintaining high-quality video frames for participating high-end workstations.
- **Failures** — A failure is a deviation of a system from its normal operation due to some errors or faults. Various kinds of failures can occur during a video-conferencing session. These failures include hardware, software, network, and many others. Among them, the most relevant failures to a video-conferencing system are the following:
 - **Site failures** — The participating site may be down for some time and re-join the session later. This could be caused by various kinds of errors from hardware, operating system, software, memory, or disk. To deal with site failure, a video-conferencing system may record the session in some permanent storage and the failed site may retrieve it later when it resumes normal operation. To prevent site failure caused by disk error, a distributed system may use redundancy storages to store a duplicate copy of the data.
 - **Network failures** — The network may be disconnected for a time or the network may be congested due to large volume of traffic. To deal with network failure, the packets may be re-routed through other connections, or the system can store the data to be sent in some temporary storage and retransmit when the network becomes available.
 - **System failures** — These failures refer to a wide range of system-limit violations. For example, the hardware may malfunction for a time. There may be too many processes running on a system, which means no additional process can be created when needed. The disk space may be full, which means no temporary file can be created and the normal processing of the system cannot be continued. To deal with system failures, the application may go into a recovery mode and wait for the system to resume its normal state, then it can keep on running on its normal mode.

- **Number of participants varies** — The number of users participating in a video-conferencing session varies. This changes the demands placed on the video-conferencing system. If the system load is too heavy, then the system may limit the number of users to provide reasonable service to each participant. It can also degrade the LoS for some users to maintain the system availability to all users.

To deal with anomalies, a system can detect and report them, it can tolerate them, or it can be designed to fix the anomalies. A system must be able to detect anomalies to avoid an inconsistent state. There are two different kinds of anomalies: those that pop up suddenly and those that occur gradually. Each can be dealt with by a different strategy. For example, a power failure can occur and cause the system to stop immediately. If anomalies occur suddenly, the system may not prepare enough to recover from it. The system may immediately stop working so it does not cause further damage. This is called a fail-stop mode. However, sometimes the failure of the system can cause cascade damages.

The distributed system should be prepared to deal with sudden anomalies and guarantee that the anomalies will not cause a lot of inconsistency to the system. For example, database logs constantly record the system status in the permanent storage. If a sudden failure occurs, then the system can recover the database back to some old but consistent state. Although some transactions may be redone or undone, the information contained in the system log can guarantee that no further damage will occur to the system.

A video storage system can record all frames for a session and then make them available to play again. If anomalies grow slowly, then the distributed system can prepare more for recovery. The system can be designed to tolerate them, maybe by degrading the service gracefully, or it can try to fix the anomalies and resume normal processing. If the system wants to tolerate the anomalies, then it may substitute or reconfigure one of its components. However, the substituted component may be inefficient, it may be reliable only under limited circumstances (inputs), or it may be operational only for a limited time. For example, if a direct link between two sites was disconnected, the system may decide to re-route the data through some indirect route, with the penalty of additional delay for the data. There are always some overheads that exist if a distributed system must prepare to deal with anomalies. For example, a checkpoint is used to help recover processing back to a consistent state in the case of some failures, but there are some overheads associated with a checkpoint due to the periodic recording of the system status.

Adaptability allows a distributed system to accommodate anomalies and reconfigure itself to provide service based on the available resources.

6.1.4 Network Video Software

The DSC team selected NV, a popular video-conferencing tool, developed by XEROX PARC, as the testbed to implement adaptability features. The team extended NV to

provide adaptability to network characteristics and to allow it to reconfigure itself to different parameters during anomalies. Being hardware-independent, the DSC team compared the overheads of NV in the software implementation of the team's algorithms without having to exclude the performance benefits due to the hardware of certain codec schemes.

6.1.4.1 Structure of Network Video

Network video transmits and receives video data across the Internet using remote transport protocol (RTP) built on top of user datagram protocol (UDP). Network video runs on a wide range of network bandwidths, and it can support slow-frame-rate video over a slow modem line, while also allowing higher quality video over a high-speed local area network (LAN). Network video uses a video-compression scheme designed to perform at a reasonable speed when implemented in software. It takes advantage of the similarity between consecutive frames in a typical video stream and the similarity between neighboring pixels in any given region of the frame. This block-based compression scheme takes only tens of milliseconds on a workstation to compute. The stationary blocks are periodically transmitted to improve image quality, which means packet losses are made up by retransmission triggered by motion.

Network video has three main modules:

- **Grabber module** grabs frames using the platform-specific video-capture board. It performs analog-to-digital conversion in hardware.
- **Encoder module** does lossy compression. Human eyes are more sensitive to luminance than chrominance. By converting the red, green, blue (RGB) model to luminance-chrominance (YUV) model (the process of subsampling), the encoder module reduces chrominance information and retains luminance information. The encoder module also executes the block-based conditional-replenishment algorithm. Haar transformation is applied to each block to pick up the high-frequency components that help provide better looking text. The encoding process is finished with dead-zone quantization, where a single threshold value is used to eliminate low-energy terms and run-length coding.
- **Sender module** transmits the compressed frames across the Internet to its client processes.

6.1.4.2 Reconfiguration of Network Video

Adaptability features of NV include color depth, frame resolution and size, and codec scheme. To incorporate adaptability into NV, important aspects must be considered, including inputs, reconfiguration policy, and reconfiguration mechanisms.

6.1.4.2.1 Inputs

To notify NV about changes in the system environment, there needs to be input from outside NV. Input parameters for adaptability that should be monitored from NV applications include number of users, delay/jitter, frame rate, color, frame size, and frame resolution. Input parameters that are available in DSC include CPU use, available network bandwidth, and storage requirements. The inputs can come from users, which means the parameters can be selected and modified interactively by users when the service is unsatisfiable. They can also come from the distributed system, which means the inputs come automatically from the distributed system based on current parameters of the environment, such as available bandwidth and CPU use.

If the input is from users, then it can be specified by selecting the LoS or by specifying actions that must be done by NV, such as compression of color depth or reduction of resolution. The input can also be some direct change to any parameters, such as color or frame size.

If the input is from the distributed system, then the distributed system must periodically measure the available resources, including network bandwidth and CPU use. Based on the currently available resources, the distributed system calculates the necessary input parameters for NV. The input parameters are the result of a decision algorithm (usually in the form of an object-function evaluation) in terms of one or more Quality of Service (QoS) parameters and application attributes, such as timeliness, precision, and accuracy. When it receives the new parameters, NV reconfigures itself to run under the new LoS.

When anomalies are detected and new available resources are measured by DSC, it must make decisions on what action should be taken by NV and notify NV of its decision (Ref. 6.1.4.2.2). Section 6.1.4.2.3 describes how the parameters and behaviors of NV will be changed when DSC chooses the policy.

6.1.4.2.2 Reconfiguration Policy

Reconfiguration policy is a decision derived from a set of decision rules. Reconfiguration policy guides the choice among several alternatives when a reconfiguration is possible and needed. In a reconfiguration policy, there are several sets of mutually exclusive alternatives that have to be traded. These trade-offs include communication/compression and timeliness, accuracy, precision (TPA).

The trade-off between communication and compression depends on the available network bandwidth and CPU use.

Table 6-1 shows the CPU use under various LoS.

If the data is compressed before sending, then it takes less bandwidth to send the data. However, it takes additional CPU time to compress the data at the sender side and to decompress it at the receiver side. If the data is sent without compression, then it takes more bandwidth to send the data but less CPU time for the sender and receiver to process it.

Figures 6-1 and 6-2 show that the percentage of time encoding and decoding over total processing time decreases as the compression ratio increases from 1:1 to 1:16. Therefore, the decision to compress data should be made based on available network bandwidth and CPU use.

Timeliness is defined as *when an event is to occur*. Maintaining it means meeting a deadline. Accuracy is defined as *the degree to which the output conforms to the semantics and contexts of the applications*. Maintaining it means guaranteeing the correctness of the data. For example, lossy-compression algorithms cause the loss of accuracy. Precision is defined as *the quantity of information provided or processed*. Maintaining it means maintaining the amount of data being processed or transmitted over the network. For example, the number of frames per session, number of pixels per

Table 6-1 CPU Use Under Various LoS.

Level	Configuration	CPU Use
1	(G, 128, S)	67 - 72
2	(G, 256, S)	76 - 81
3	(G, 512, S)	78 - 83
4	(C, 128, S)	64 - 69
5	(C, 256, S)	04 - 79
6	(C, 512, S)	77 - 82
7	(G, 128, M)	80 - 85
8	(G, 256, M)	86 - 90
9	(G, 512, M)	89 - 93
10	(C, 128, M)	78 - 84
11	(C, 256, M)	86 - 89
12	(C, 512, M)	91 - 94
13	(G, 128, L)	92 - 97
14	(G, 256, L)	95 - 98
15	(G, 512, L)	95 - 98
16	(C, 128, L)	95 - 97
17	(C, 256, L)	95 - 97
18	(C, 512, L)	95 - 97

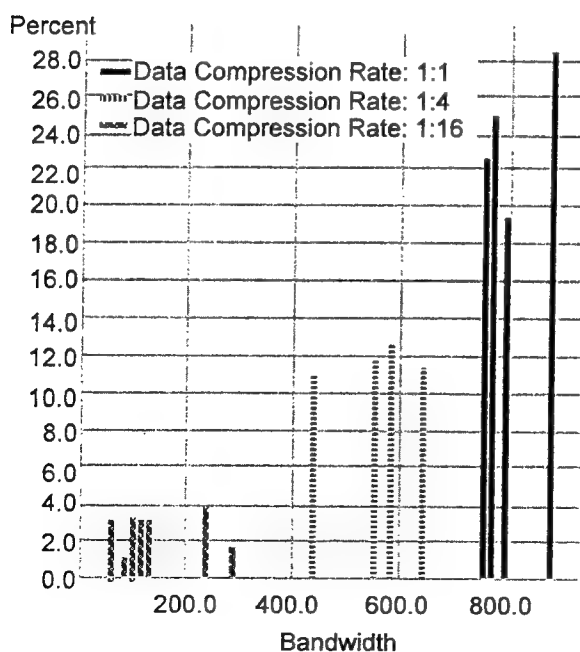


Figure 6-1. Percentage of time encoding over total processing time for resizing.

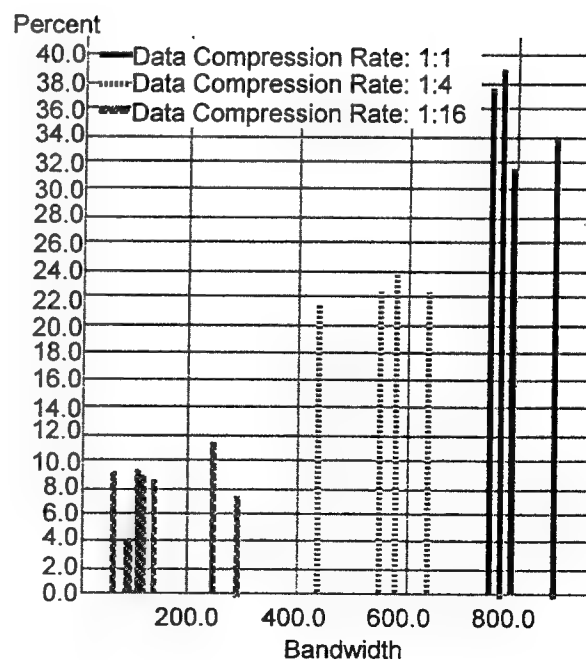


Figure 6-2. Percentage of time decoding over total processing time for resizing.

frame, and number of bits per pixel are parameters used to describe the precision of a video-conferencing session. Timeliness, precision, and accuracy cannot be

simultaneously maintained at the highest level during anomalies. The user must trade-off among these attribute values. The policy to trade-off is the following:

- Maintaining timeliness when bandwidth decreases:
 - Reduce frame size. The accuracy is maintained unless the frame size is below a certain value.
 - Reduce frame resolution. The accuracy and precision are reduced.
 - Dither color frame to black and white.
 - Compress color depth.
 - Switch to a codec scheme that has a higher compression ratio; side effect: CPU use increases, which can be compensated for by frame resizing and resolution reduction.
- Maintaining accuracy when bandwidth decreases:
 - Switch to a lossless codec scheme with reduced frame size.
 - Dither color frame to black and white.
 - Compress color depth. Compress Y and UV no more than 2 bits each.
 - Do not use lossy codec schemes.
 - Do not reduce frame size or resolution by a big factor.
- Maintaining timeliness when CPU use increases:
 - Switch to a codec scheme that requires less computation, usually with lower compression ratio.
 - Reduce frame size.
 - Dither color frame to black and white.
 - Do not compress color depth.
 - Do not reduce frame resolution.
- Maintaining accuracy when CPU use increases:
 - Switch to a lossless codec scheme.
 - Reduce frame size.
 - Dither color frame to black and white.
 - Do not compress color depth.
 - Do not reduce frame resolution.
 - Do not use lossy codec schemes.

6.1.4.2.3 Reconfiguration Mechanisms

Reconfiguration mechanisms refer to the implementation of the reconfiguration policy that has already been decided. It is called the infrastructure of the adaptability in a system. For the DSC infrastructure, the team extended NV by adding new modules and modifying the three modules that existed in the original design. Figure 6-3 shows the extended architecture. The new modules added to NV are the network-probe module, admission-control module, and video-conferencing adaptability collaborator module.

The network-probe module constantly monitors the network. It is integrated with the Wide Area Network Communication Emulation (WANCE) tool to track round-trip time and routing information of the packets. The admission-control module restricts the number of users if there are insufficient resources available. The video-conferencing adaptability collaborator module includes three new modules:

- **Recorder** stores video data frames for future retrieval, retransmission, and reference.

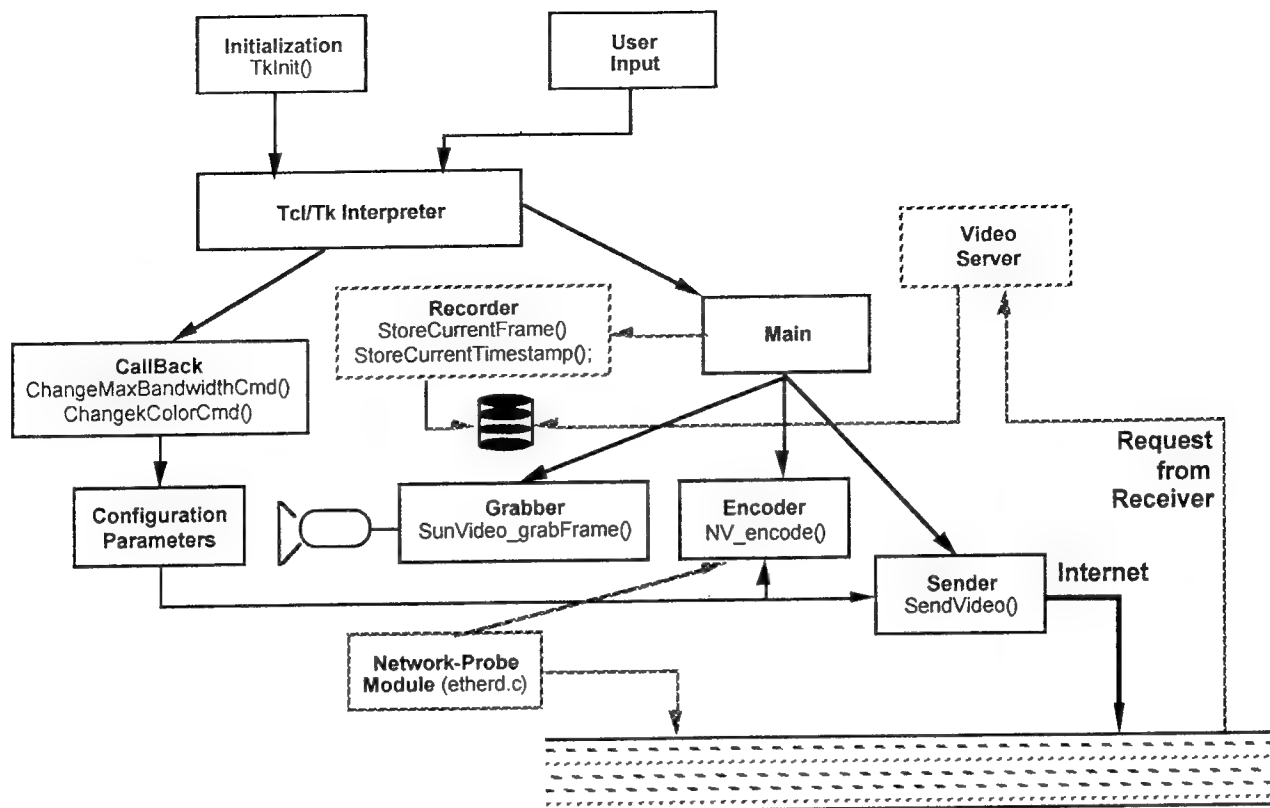


Figure 6-3. Extended architecture of network video.

- **White board** provides a way to write and send text, graphs, and other material during a conference.

In addition to the new modules, the team modified the three original modules:

- **Grabber:**
 - Change sampling rate, which is the frequency of grabbing frames in a stream of input. For example, grab alternate frames instead of all of them.
 - Reduce load on the sender machine to save computation in the encoder module.
 - Reduce load to help sender receive better (packets will not be lost).
- **Encoder:**
 - Incorporate switching between Haar and Discrete Cosine Transform (DCT) transformations, depending on the application. Haar is good for video frames with high frequency components, such as edges. The DCT is good for other kind of frames, such as movie frames.
 - Add the ability to encode different levels of color: Full color is 24 bits per pixel (YUV); greyscale is 8 bits per pixel.
 - Extract the 16 UV bits.
 - Perform 4:1 reduction.
 - Merge the new 4 UV bits with the original 8 Y bits to get a total of 12 bits; similar to the dithering procedure.
 - Change resolution of frames.
 - Compute the value of a pixel as an average of 4 pixels (Figure 6-4).
 - Can be coded for Sun Video card.

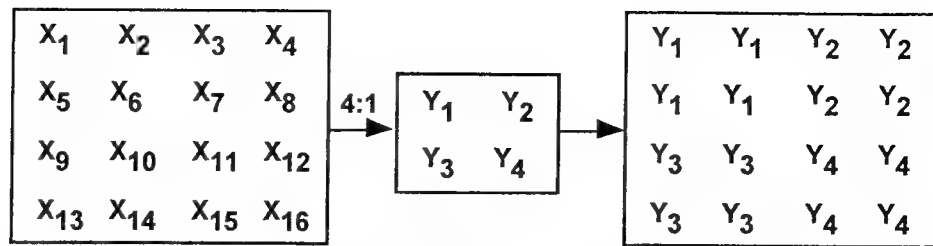


Figure 6-4. Frame compression using pixel averaging to achieve 4:1 reduction.

- **Sender:**

- Switch protocols, depending on current network conditions, to achieve reliable transmission, better flow control, and better jitter control.
- Prioritize sessions:
 - Add callback function (SetPriority) so that users can specify priority at the beginning of a session (static).
 - Add callback function (ChangePriority) so that users can modify a priority when a session is in progress (dynamic).
 - When network is congested, lower priority sessions are stored and not transmitted.
- Admission control restricts the number of users in a session.

The network-probe module monitors the network constantly. It measures network traffic, detects congestion, and calculates available bandwidth. It interfaces with the sender and encoder modules through shared memory. The variable bandwidth can be accessed by the grabber, encoder, and sender modules. There are two domains in monitoring the Internet. The network-probe module can either monitor the local network up to the gateway, or it can go beyond the gateway. However, statistics gathered beyond the gateway might not be accurate because the route the packets will take is unknown. In the DSC implementation, the team integrated the network-probe module with the WANCE tool to get a better environment for the experiment.

The admission-control module restricts the number of participants per session if sufficient resources are unavailable. It stores the session and retrieves them for later playback for users unable to join a session. It prioritizes sessions, media, recipients, and messages. It will cut conversation time or close the conversation for sessions with lower priorities, and send the most important messages first if the conversation time is limited. It closes the less important medium to save bandwidth when the available bandwidth decreases. In case of site failures, network congestion, or extreme low bandwidth, the admission-control module uses the video-conferencing collaborator to record a video session for later playback, retransmission, and retrieval.

The recorder module records video frames in the video-stream database at the sender and receiver machines. The functionalities include the following:

- Record frames every fixed interval (automatic, inflexible).
- Record the changed part (difference, mostly foreground).
- Record frames in variable rates (based on relative importance).
- Record frames based on user-defined criteria (e.g., some particular time intervals) or frame features (e.g., some particular scenes).

The summarization module performs user-aided extraction of significant frames to organize frames by theme. It includes off-line processing because it is time-consuming and hard to achieve realtime for typical sessions. It includes on-line, user-assist processing for slowly evolving scenes. However, to effectively summarize a session, some computer-aided summarization (CAS) tools are needed. In addition to video, audio provides more information and can be combined into the summarization module for processing.

The white-board module provides a user interface to NV. The module lets users see simultaneously the speaker and other information, such as slides, graphs, images, and maps. However, using the same window for speaker and information generates a lot of traffic in NV because of background changes. In the DSC design, the sender and receiver can interactively point to and modify objects on the white board. In DSC, user inputs are parsed using the Tcl/Tk interface and grabbed from the screen using the grabber module. There is also an interface with the video compression collaborator in our implementations.

6.1.5 Experiments and Results

6.1.5.1 Experimental Setup

The team chose Network Video (NV) and enhanced it to incorporate adaptability and recording features, then used it as a testbed to conduct performance studies.

The platforms for the experiments included a Sun Sparc 10 station and a Sun Sparc 5 station connected in a LAN, and two video cameras. The workstations ran the Solaris 2.3 operating system.

6.1.5.2 Observations and Discussions

6.1.5.2.1 Experiments on Color Depth Compression

Two color models are often used in practice: RGB and YUV. Y stands for the luminance component of a pixel; U and V represent the chrominance components of a pixel. Many systems use the YUV model because it is easy to adjust colors. For example, in the YUV model, UV components can be changed easily and separately without disturbing Y components and vice versa. In the NV system, frames represented in RGB space are first captured and then converted into those in YUV space. This process is called subsampling. Because human eyes are less sensitive to chrominance than to luminance, the team can reduce the number of bits in encoding U and V components. They can do this while retaining the number of bits in encoding Y components and still guarantee relatively good quality in decoded color frames. Nevertheless, subsampling is lossy

Although the team can achieve different levels of colors by using a different number of bits to encode UV components, they may still be needed to reduce the number of bits in encoding UV when the bandwidth is too small and the number of bits in encoding Y cannot be reduced any further.

In an NV system, Y and UV components of a pixel are each 8 bits long. Color-depth compression is achieved by using fewer bits to represent Y and UV components, respectively. The team experimented with various configurations set up by separately compressing Y and UV components each by 1, 2, and 4 bits. For example,

compressing UV components by 2 bits means they are represented with only 6 bits. Table 6-II shows the effect of reducing the number of bits in UV and Y components of a pixel on the quality of a video frame. One may divide Table 6-II into three regions: A, B, and C, as described below:

Table 6-II. Image Qualities for Different Compression Factors for Y and UV Components.

UV \ Y	0	1	2	4
0	original	good	good	noisy
1	snowy	good	good	noisy
2	less snowy	snowy	good	noisy
4	with grid	with grid	less snowy	noisy

Region A is indicated by an arrow pointing to the top-left 2x2 area. Region B is indicated by an arrow pointing to the bottom-left 2x2 area. Region C is indicated by an arrow pointing to the rightmost 2x2 area.

- **Region A** — Very good image quality can be achieved when either Y components alone or both Y and UV components are compressed by a small number of bits (up to 2 bits). Thus, up to 4 bits of combined Y and UV component compression can be achieved when a very good image quality is being maintained.
- **Region B** — Human eyes can accept image quality in this region when the number of bits compressed for Y components is less than 4 bits. This can be attributed to more compression in UV components than in Y components. Thus, more bits in encoding UV components can be reduced than those in encoding Y components, as long as the compression for Y components does not reach a threshold value (4 bits).
- **Region C** — The image quality in this region drops dramatically (though the image is still recognizable) when the number of bits compressed for Y components reaches a threshold value (4 bits in this case). Thus, the team has to compress more bits for UV components if the bandwidth becomes too small because the number of bits in encoding Y cannot be reduced any further.

Therefore, the conclusions are the following:

- Stay in Region A and obtain very good image quality if the combined number of bits of compression for Y and UV components of a pixel is not expected to exceed 4 bits.
- Enter Region B and obtain not-so-bad image quality when the combined number

of bits of compression for Y and UV components of a pixel is greater than 4 bits but is no larger than 6 bits.

- Enter Region C for a higher compression rate in combined number of bits to encode Y and UV components of a pixel but image quality is degraded substantially. Due to the relative importance of luminance components, the team should not compress Y for more than 4 bits if it wants to maintain a recognizable image quality.

6.1.5.1.2.2 Experiments on Resolution Reduction and Frame Resizing

To determine how frame resizing can be used for adaptability for video conferencing, the DSC team conducted experiments to measure the frame rates for frames of different sizes (Figure 6-5). For a particular frame size, as the available network bandwidth decreases, the corresponding frame rate also decreases. This results in loss of continuity and smoothness of video presentation at the sender and remote sites. In such a situation, the video-conferencing system adapts by changing to a smaller frame size to maintain (or even improve) the original frame rate. The DSC system supports only four discrete levels of frame sizes. Thus, the frame rate may be changed (improved) when the system changes to an LoS with less bandwidth. In the future, the team plans to provide more levels of frame sizes, which will allow the system to adhere closely to the current operating frame rate while reducing the network bandwidth requirements at the same time.

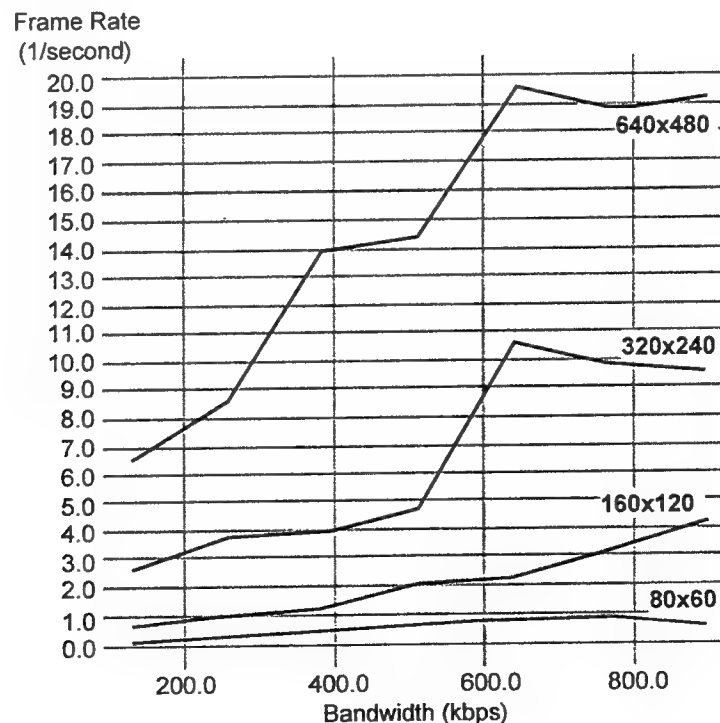


Figure 6-5. Frame rates for resized frames.

To implement the dynamic frame resizing and resolution reduction, the team manipulated data before it was encoded and after it was decoded. Though it takes extra CPU time, the results show that this overhead was tolerable. The team computed the average time percentages for encoding and decoding in processing one frame. In

frame resizing, the combined time percentages for encoding and decoding were more than 50 percent of the overall processing time of one frame, which meant that encoding and decoding were the most expensive parts in video conferencing.

When the compression factor value became 2 or 4, the combined time percentages for encoding and decoding dropped dramatically; they were no longer the most expensive parts of processing in video conferencing (Figures 6-6 and 6-7). Instead, the video transmission became the most expensive part of processing in video conferencing. Similarly, in resolution reduction, the time percentage for encoding decreased when the frame size decreased (Figure 6-8). However, the time percentage for decoding when the compression factor was equal to 2 or 4 increased compared to that when the compression factor was 1 (Figure 6-9). This was due to the extra computation overhead involved in resuming the original frame size. But the combined time percentages of encoding and decoding over total processing time were only slightly larger than those for original NV because of the decrease in the amount of data when the network was busy.

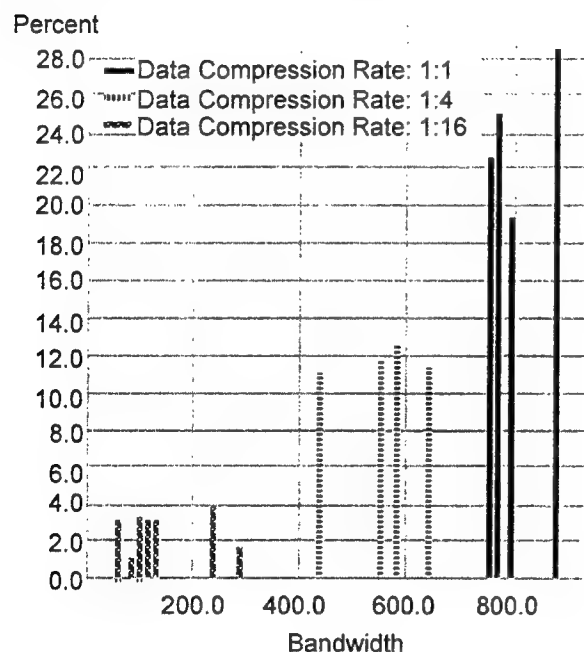


Figure 6-6. Percentage of time encoding over total processing time for resizing.

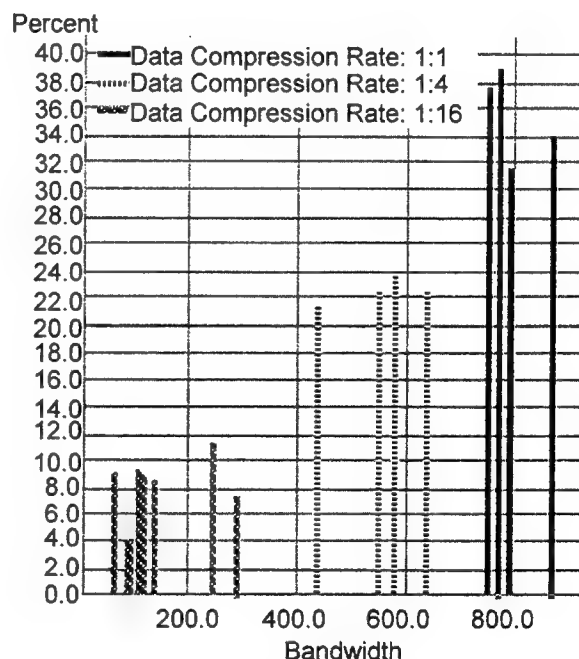


Figure 6-7. Percentage of time decoding over total processing time for resizing.

6.1.5.1.2.3 Experiments on Codec Schemes

The original NV software supports three codec schemes: native NV scheme, CellB scheme invented at Sun Microsystems, Inc., and CU-SeeMe scheme. The DSC modified version of NV supports a scheme that replaces the Harr transform in the native NV scheme with the DCT algorithm. The DCT is the core transform used in many codec schemes, including JPEG and MPEG. The team implemented the DCT algorithm in NV and studied the performance of the four codec schemes in the context of NV.

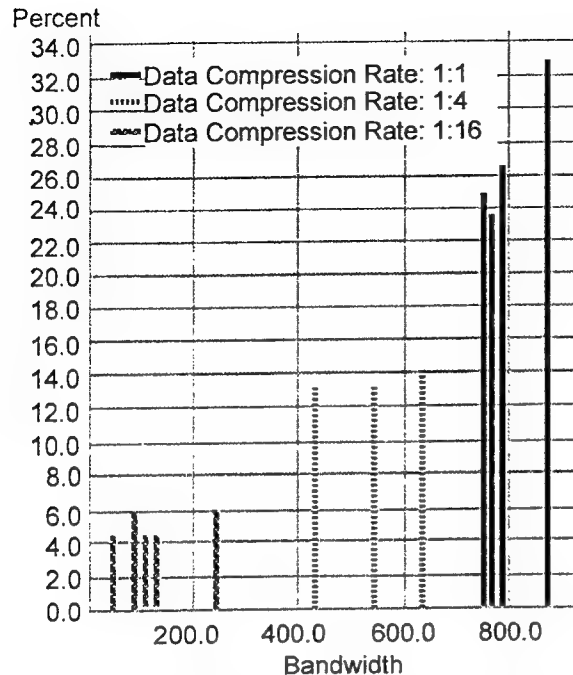


Figure 6-8. Percentage of time encoding over total processing time for resolution reduction.

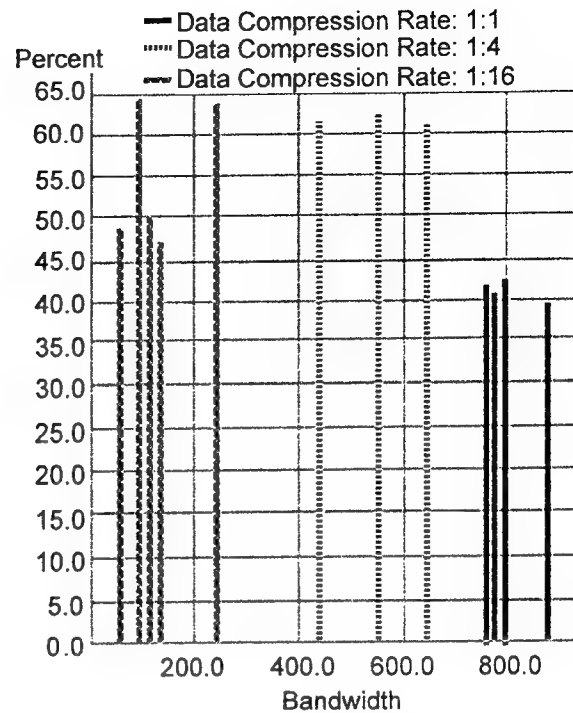


Figure 6-9. Percentage of time decoding over total processing time for resolution reduction.

The native NV compression scheme is a sequence of Harr transform, quantization, and run-length coding. The uncompression process is the inverse. The team replaced the Harr transform with the DCT algorithm in the compression process and replaced the Harr inverse transform with the inverse DCT in the uncompression process.

The importance of the incoming video data from the camera has been ignored to a certain degree in the performance study of video-conferencing softwares. The team investigated this in some detail and found that CPU use and frame rate also depended on the amount of motion in each frame and its frequency distribution. The latter depended on the content of a frame. For example, a textual frame usually has more high-frequency components, whereas a continuously toned frame has more low-frequency components.

To measure the amount of motion precisely, the team defined the momentum of a frame to be the number of blocks whose contents had been changed. The team defined the relative momentum of a frame to be the percentage of the number of blocks whose contents had been changed over the total number of blocks in that frame. The team's hypothesis was that the processing time for video data was heavily influenced by the momentum. The DSC experiments supported this assertion.

The team also conducted a series of experiments to study the performance of NV systems under various configurations. The configurable parameters included codec schemes (native NV scheme, CellB scheme, DCT scheme, and CU-SeeMe scheme); frame sizes (160 x 120 (small), 320 x 240 (medium), or 640 x 480 (large); and display

modes (color or greyscale mode). It measured CPU use of NV under various configurations; see Table 6-III.

Table 6-III. CPU Use in Various Codec Schemes Supported by NV.

Coding Scheme	Size	Grey Scale			Full Color		
		Small	Medium	Large	Small	Medium	Large
Native NV Scheme		[62, 67]	[72, 80]	[99, 103]	[61, 66]	[71, 77]	[100, 103]
DCT Scheme		[63, 67]	[74, 75]	[103, 108]	[60, 63]	[70, 73]	[101, 105]
CellB Scheme		[4.6, 6.5]	[4.9, 6.5]	[4.3, 7.6]	[5.5, 7.4]	[6.4, 7.4]	[6.5, 9.4]
CU-SeeMe Scheme		[66, 70]	[73, 76]	[99, 101]	Color mode not supported		

Note that in many cases, displaying frames in greyscale mode took more CPU time than in color mode. This appeared to contradict the fact that displaying frames in color mode was more expensive than in greyscale. However, it implied that the frame rate in greyscale mode was higher than that in color. Table 6-IV shows the frame rates under the same configurations.

Table 6-IV. Frame Rate Obtained Using Various Codec Schemes Supported by NV.

Coding Scheme	Size	Grey Scale			Full Color		
		Small	Medium	Large	Small	Medium	Large
Native NV Scheme		1.9	0.5	0.2	1.6	0.4	0.2
DCT Scheme		1.9	0.8	0.3	0.7	0.2	0.1
CellB Scheme		6.0	1.5	0.3	6.0	1.6	0.3
CU-SeeMe Scheme		5.9	1.0	0.8	Color mode not supported		

Figures 6-10 and 6-11 show how the distribution of the time spent in forward Harr transform and DCT for a frame depends on the relative momentum of that frame. Figures 6-12 and 6-13 show that for corresponding reverse transforms, the bigger the relative momentum, the longer a transform lasts.

The team compared the performance between the NV scheme and the DCT scheme. Figures 6-14 and 6-15 show that the forward and inverse transforms of DCT take more time than those of NV because cosine functions in DCT are transcendent, and Harr functions are elementary, which implies that DCT is more computationally expensive than Harr transform. But the compression ratio is not in the realm of the team's expectation. Figure 6-16 shows that the compression ratio of NV is about twice that of DCT. This test was done for continuously tuned frames. DCT should have better compression ratio than Harr transform for such frames.

6.1.5.1.2.4 Experiments on Session Recording

Session recording involves both the computation overheads and secondary storage requirements. One important parameter to consider is the recording frequency, which

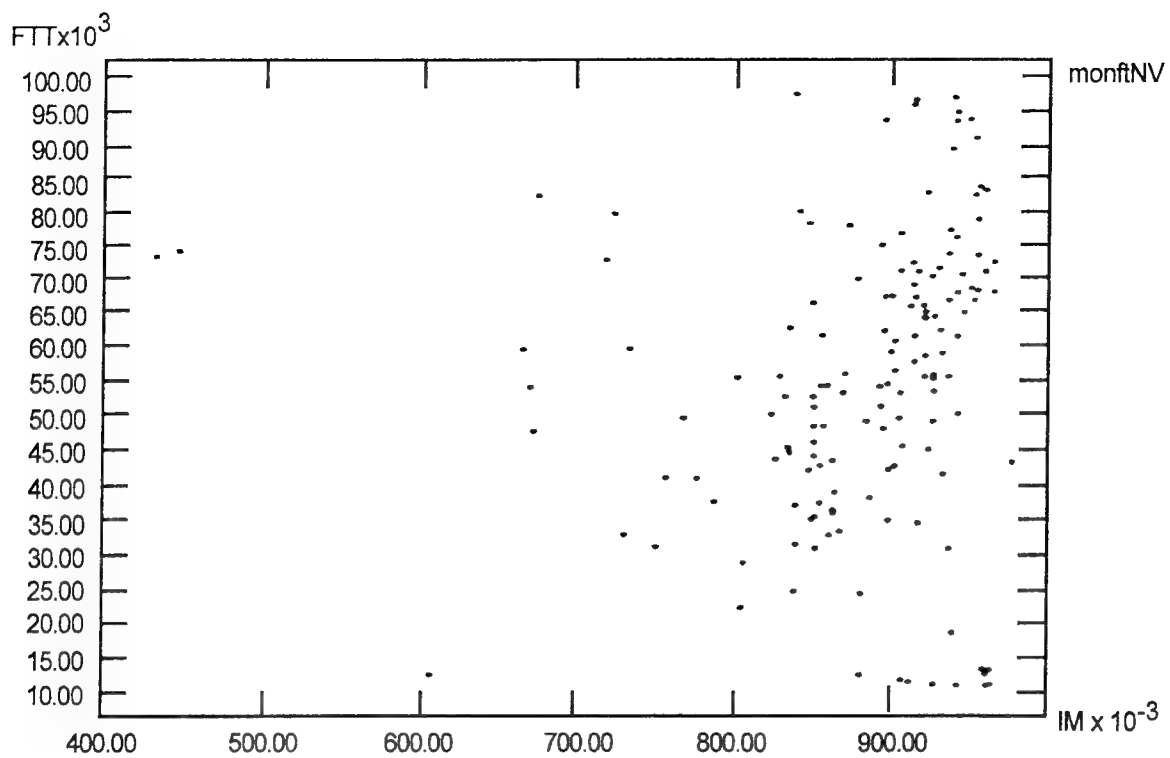


Figure 6-10. Impact of momentum on DCT forward transform time.

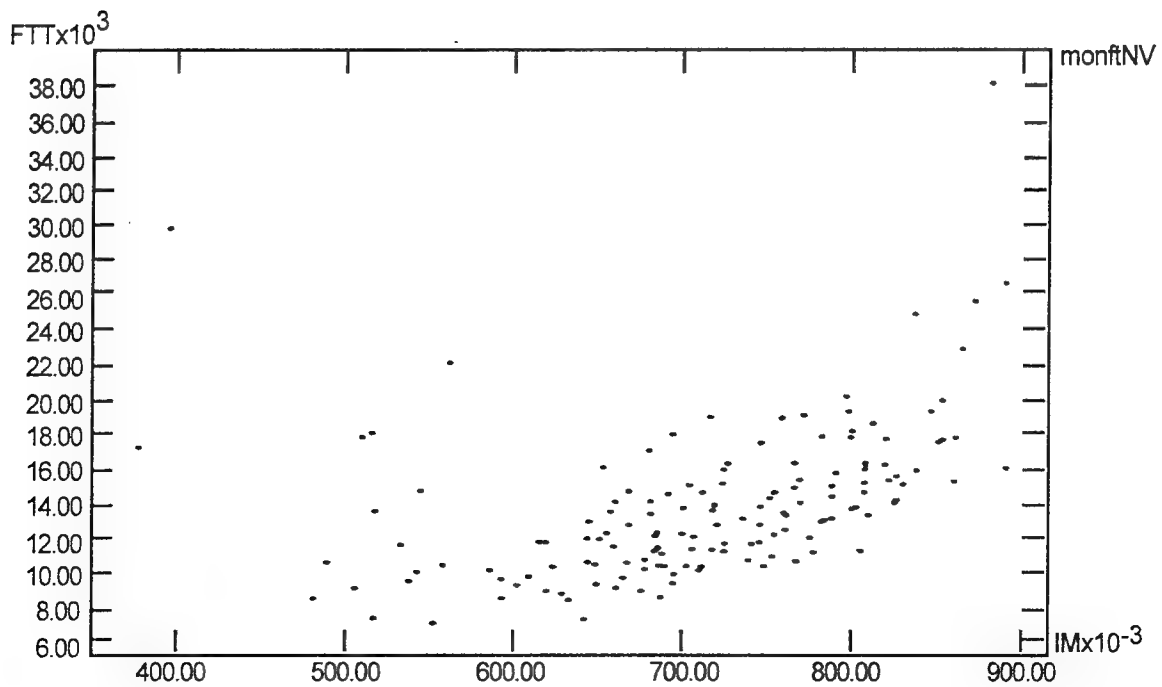


Figure 6-11. Impact of momentum on NV forward transform time.

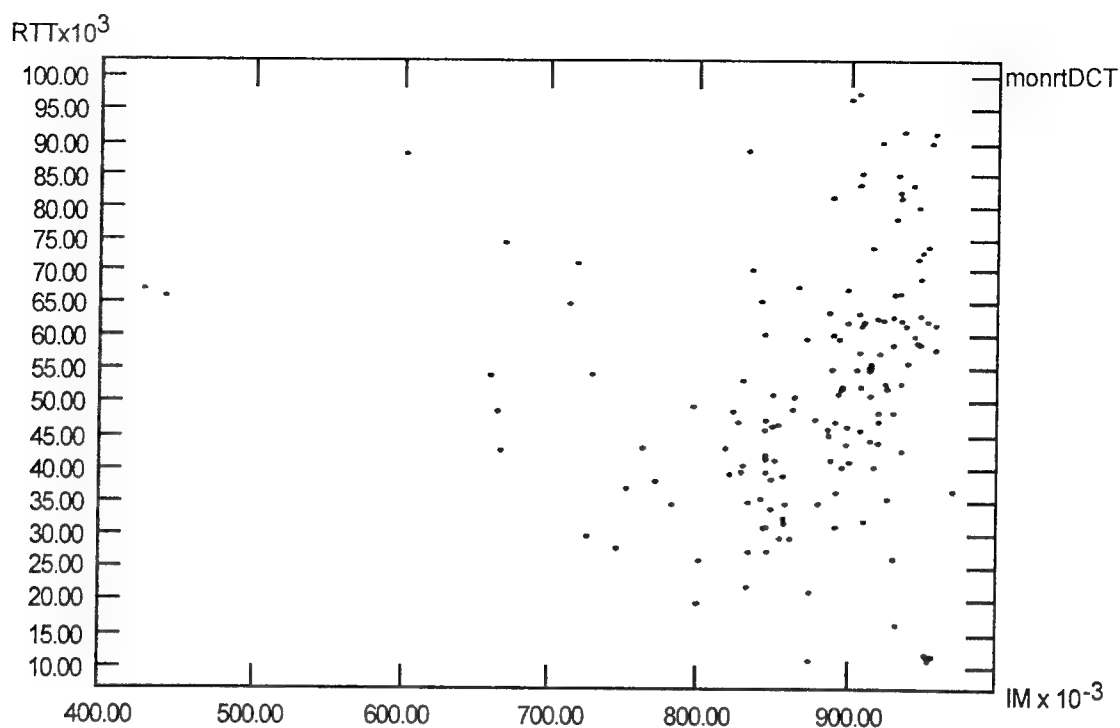


Figure 6-12. Impact of momentum on DCT reverse transform time.

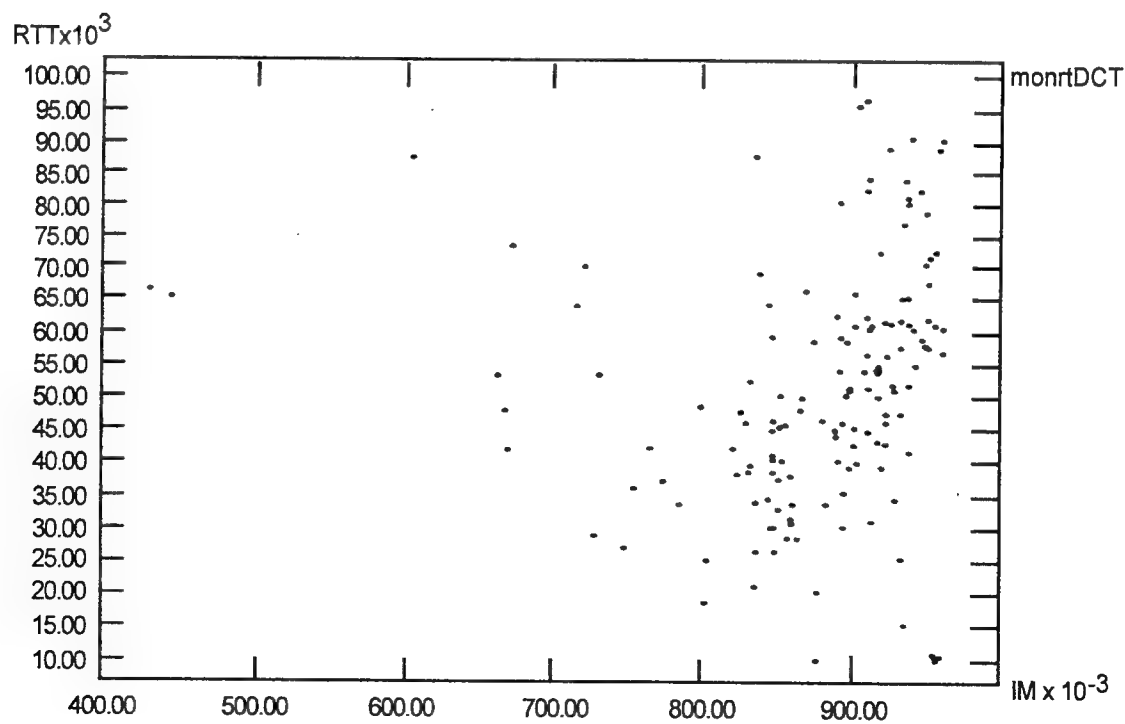


Figure 6-13. Impact of momentum on NV reverse transform time.

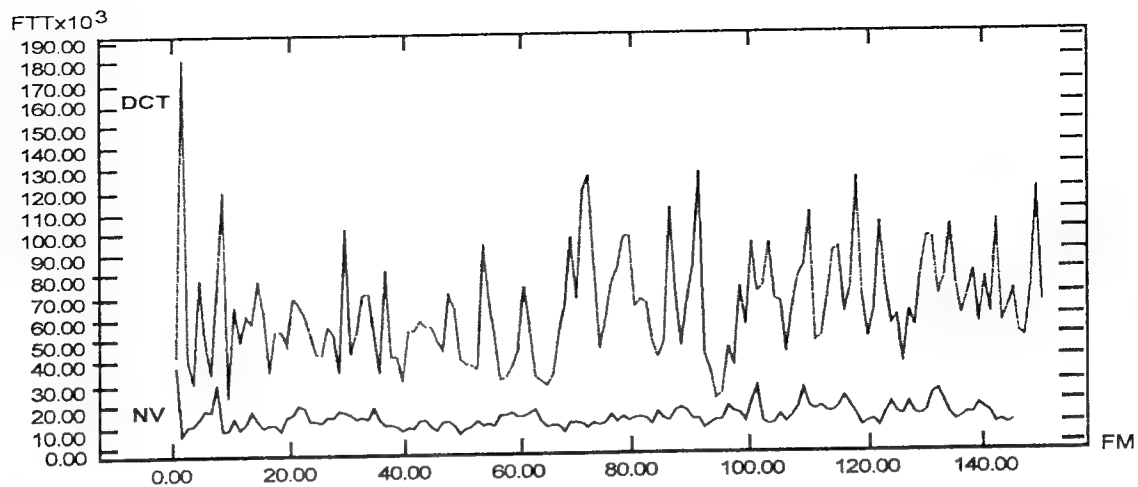


Figure 6-14. Time spent in forward transform: NV versus DCT.

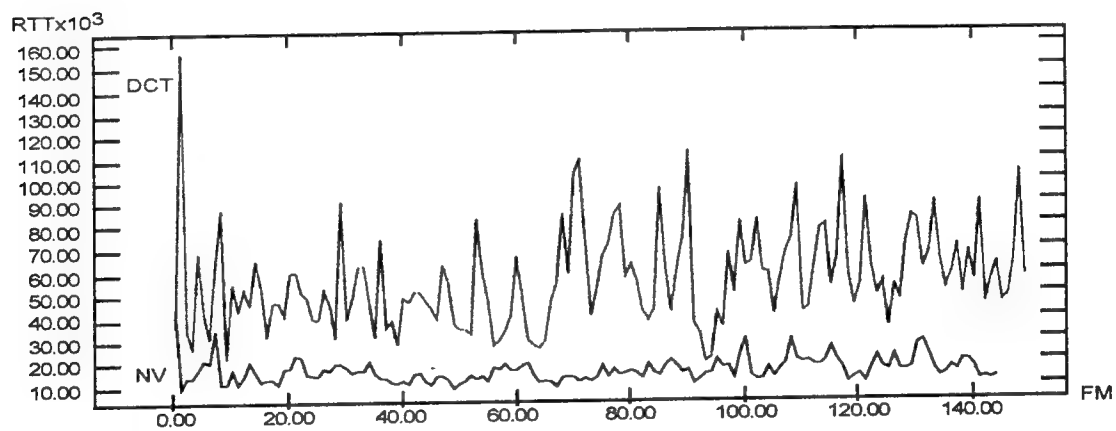


Figure 6-15. Time spent in reverse transform: NV versus DCT.

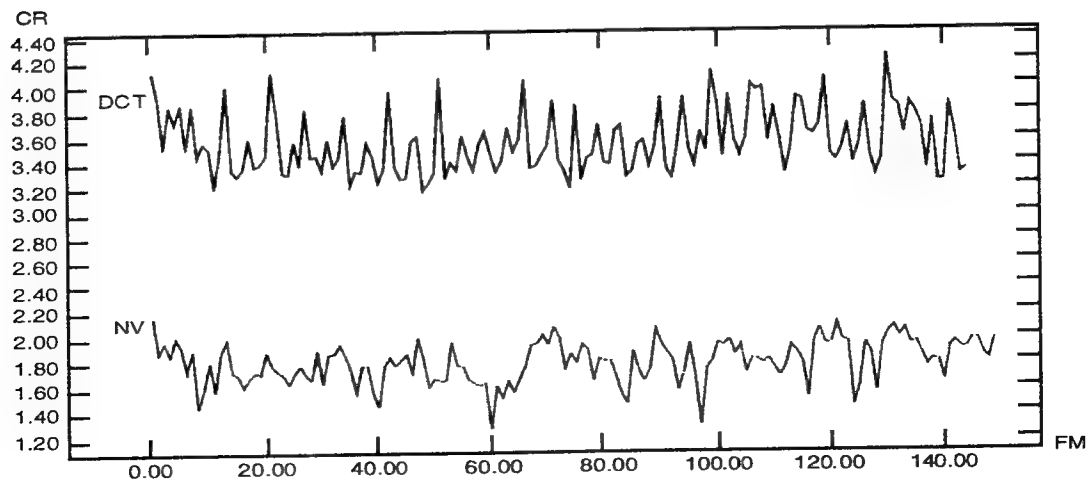


Figure 6-16. Compression ratio: NV versus DCT.

is the frequency at which the video frames are stored. This must be differentiated from the display frequency, which is the frequency at which the video frames are updated on a user terminal. The latter determines the continuity and smoothness of the video-session presentation. Storing every video frame generated during a video session is sometimes unnecessary because many consecutive frames may have almost the same contents, and because it would consume a lot of disk space. Conversely, recording frames at a very low recording frequency may hurt the smoothness of video sessions and degrade the video quality during replay.

Recording video frames imposes computational overheads that can affect the rate at which video frames are generated for display (i.e., the display frequency). Because the system records the video-conferencing sessions only when an anomaly occurs — for example, when the available network bandwidth decreases dramatically — imposing computation and network overheads may affect the frame rate (display frequency) even more significantly. Figure 6-17 shows the comparison between the frame rates without recording and the frame rates with recording (recording every frame) under different available network bandwidths. The overhead (signified by drop in frame rates) imposed by recording the whole session is very small when the available bandwidth is small. This means that recording the entire session does not contribute to a large performance degradation when network bandwidth becomes a bottleneck. This is counter intuitive to the team's reasoning presented earlier. However, this can be explained as follows: When the network bandwidth is low, the system generates/transmits fewer frames per second to remote recipients. The low frame rate results in residual computation cycles on the system that are under used. These computation cycles can be used for storing the frames in the database without affecting the frame rate. Conversely, high available network bandwidth allows the system to generate/transmit larger number of frames to remote recipients. The process for storing the frames now must compete with the process for generating/displaying/-transmitting the video frames, which resulted in a drop in frame rate to almost half of the original.

The team is currently developing support for storing full-quality video frames at the sender site. It also makes as a parameter the frequency at which the frames must be recorded at the local disk. This parameter can be specified by the sender at run-time. The team is experimenting in finding a frequency optimal for both the storage and the video quality.

Another concern in recording video sessions is the storage-bandwidth requirements for recording the video frames. It is possible not to record every frame into the database and still maintain the quality of video replay. The team has observed the effects of changing recording frequencies on sizes of recorded video sessions (Figure 6-18). As the recording frequency decreases, there is an exponential decay in size of the stored VC-session data. As the available network bandwidth increases, the rate at which the frame can be delivered to the remote sites also increases. Therefore, more frames are stored with the same recording frequency (time interval), thereby increasing the size of the recordings. The recording frequency takes the form of the time interval between two consecutive frames to be stored; it is actually the reciprocal of the time interval.

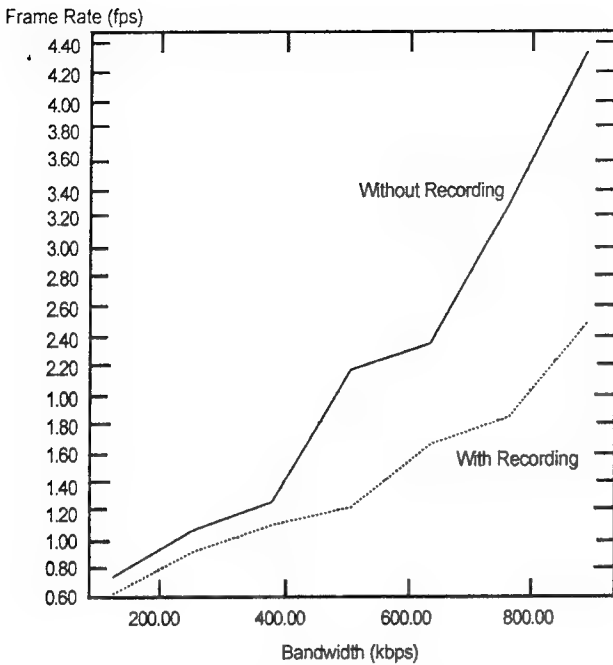


Figure 6-17. Effect of recording overhead on frame rates.

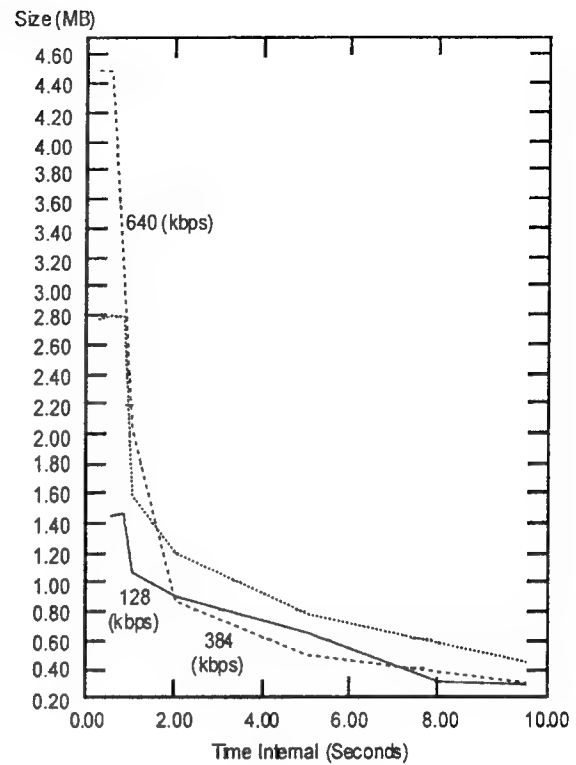


Figure 6-18. Effect of changing recording frequencies on sizes of recordings.

For a particular network bandwidth, as the recording frequency decreases, the sizes of the recorded video-conferencing sessions decay exponentially. The reason that the decay in size of recorded video sessions is almost always better than the linear decay in size is because the encoded NV video frames are not constant in size. This is because the encoding scheme in NV encodes only the differential among the consecutive physical frames grabbed by the hardware and not the complete frames.

For a particular network bandwidth, the size of the stored video session is constant up to a certain value of time interval for recording frames (recording frequency), beyond which it falls exponentially as the time interval increases between consecutive frames to be stored. The team calls this a turning point for a particular network bandwidth d_r . For time interval less than d_r , the recording is higher than the display frequency, i.e., the frames are to be stored at a rate greater than those at which they are generated. This will result in duplicates of a frame being stored. However, in the DSC implementation of the recording module, one frame is stored only once, but every frame is stored up to d_r time interval. This is achieved by storing the unique timestamp of the encoded frame in the database, and storing a frame only if its timestamp is different from that of the most recently stored frame.

To summarize, each frame is stored without duplicates for any time interval between two frames to be stored less than d_r . Beyond this point, the recording frequency is smaller than the display frequency; hence, a subset of the generated frames is stored in the database.

6.2 Distributed System Control Algorithms

This section describes the two algorithms that the Local Manager (LM) used to determine resource allocation and to measure resource use: Local Manager Allocation algorithm and the Resource Measurement and Exponentially Moving Averages (EMA) algorithm.

6.2.1 Algorithm: Local Manager Allocation

The LM performed this algorithm during every measurement period to see what processes would run and consume available local resources.

6.2.1.1 Available Information

The LM had the following data available to it when deciding which services to provide:

- Value of each LoS specified by the user on a common scale, e.g., 0 - 100: To start, the system reads a default value from a configuration file and allows the user to change the value by using a sliding scale on the LM's Graphical User Interface (GUI).
- Available resources:
 - CPU cycles
 - Memory
 - Network bandwidth (at a global level).
- Resource requirements for each LoS for the following uses:
 - CPU
 - Memory
 - Network.
- A weighted cost function for each LoS, which is the weighted sum of all required resources. The user-supplied weighted costs can be optionally multiplied by the currently measured proportion to which each resource is being used in the system. If the user selects this option, then resources that are highly available become weighted more lightly, while heavily used resources become more heavily weighted.

The system computed the weighted cost as follows:

$$\text{Cost} = (W1 * R1 * U1) + (W2 * R2 * U2) + (W3 * R3 * U3)$$

Where:

- W is weight
- R is a normalized resource-use figure for a particular service
- U is a fraction of the corresponding resources that the system is currently using. The purpose of U is to weight more heavily a resource that is less available; whereas, the purpose of W is to allow either users or system administrators to weight one resource over another.

When the system divides the value of a service by its weighted cost, it has a common metric, value-to-cost ratio, by which to compare different services.

6.2.1.1 The Problem of Maximizing Value

There is a problem of maximizing the total value of all the services provided, given the weighting criteria in section 6.2.1 and given that aggregate resource requirements for

all services on a particular host cannot exceed available resources. If the problem were limited to one available resource and if the value of each service were equated with its resource requirement, then a subset-sums problem² would result. Because the subset-sums problem, which is NP-complete, is equivalent to a simplified version of the value maximization problem, the exact solution is at least NP-hard. Section 6.2.1.2 presents a heuristic that runs in $O(n^2)$ time in the worst case and in $O(n * \log(n))$ time in the average case; n is the number of LoS. It will yield the optimal solution with few exceptions.

6.2.1.2 The Heuristic

The LM used two sorted lists of services, where the sorting criterion is the value-to-cost ratio of the service. One list was active while the other was inactive. Each list contained the currently active and inactive services, respectively.

After the LM took resource measurements, it ran the following heuristic written in psuedo code to determine if services should be activated or deactivated:

MAXIMIZE (active list, inactive list, resource availability levels).
BEGIN MAXIMIZE.

Note

Whenever the system moved a service from inactive to active, it subtracted the algorithm's measured resource use from the resource availability levels. Conversely, when the system moved the service from active to inactive, it added the service's resource use.

- Move all services in the active list to the inactive list.
- Activate the highest ratio LoS by going through the sorted inactive list from highest to lowest ratio. Pick the highest ratio LoS for each requested service. If there are sufficient resources available to accommodate it, then move it to the active list. The active list now contains the most efficient LoS in terms of the value provided per resources consumed.
- Upgrade services if there are sufficient resources by going through the inactive list from highest to lowest ratio. For each LoS, if there are sufficient resources available to accommodate them and if the LoS have higher values than the active levels corresponding to their processes, then move the inactive LoS to the active array and the active LoS to the inactive array.

END MAXIMIZE

After the Maximize algorithm completes execution, another function implements the prescribed changes by starting, stopping, or changing the LoS of affected processes. When a new process is requested, its associated services are inserted into the inactive list. After running Maximize, the system will determined if its LoS can be provided. When a user no longer needs a service, a function removes the associated LoS from the active and inactive lists and it kills the associated process.

²The subset-sum problem consists of finding a subset of a set of integers the sum of which is maximal but less than some specified integer.

This algorithm maximizes the value of the services with one exception: If the system fully uses all resources and a service is requested that requires more resources than are available, then the algorithm will not displace existing services if they have higher value/cost ratios. This is the case even if a newly added lower ratio (new service) would result in a higher aggregate value.

6.2.1.3 Example

The following is a hypothetical situation to which the algorithm could be applied:

- See Table 6-V: To simplify the example, bypass calculating a weighted cost and measuring individual resources by using a single generic resource. Suppose there are 100 units of resources and values range between 1 to 100.

Table 6-V. Generic Resource Values

Service	Value	Cost	Ratio
Video			
<video color 30>	80	80	1
<video color 15>	70	60	1.16
<video bw 30>	60	50	1.2
<video bw 15>	50	40	1.25
Audio			
<audio 3>	80	50	1.6
<audio 2>	70	40	1.75
<audio 1>	60	30	2
Drawing			
<draw 3>	30	30	1
<draw 2>	20	20	1
<draw 1>	8	4	2

- **Assume a completely unloaded system** — Enter all LoS for video into the inactive list and Maximize() selects <video bw 15> in its second step. In its third step, the video application is step by step, from highest to lowest ratio, upgraded to the highest LoS, <video color 30>, which uses 80 of 100 units.
- **Add audio capability** — After LoS are inserted into the inactive list, Maximize moves all active services to the inactive list before it moves <audio 1> and <video bw 15> to the active list. Because there are sufficient resources, Maximize() upgrades to <audio 3> and <video bw 30>, which uses 100 units of resources, yielding a value of 140.
- **Add drawing services** — Here is an example of the above mentioned case in which the heuristic does not yield an optimal solution. Maximize() puts <draw 1>, <audio 2>, and <video bw 15> in the active list. Because there are sufficient resources, Maximize() performs the following sequence of upgrades: <audio 1> to <audio 2> and <audio 2> to <audio 3>. In this configuration, the active list consists of <draw 1>, <audio 3>, and <video bw 15>, which uses 138 units, yielding a value of 94. Although Maximize() does achieve a higher overall value-

to-cost ratio, because the system has not fully used all available resources, the algorithm does not maximize the aggregate value of active LoS.

Note that all LoS for all services compete independently for execution privileges. However, levels from the same service, like video, are mutually exclusive. So, when video service is requested, only one of the levels will run. If the user requested another instance of video service, then the system would consider it separately, and it may assign different values associated with the video's LoS.

6.2.1.4 Special Case for Distributed or Client/Server Applications

There is a special case to be concerned about when the application is distributed across different hosts. If a process was unable to receive the necessary resources on its host, then the entire application would not run. This is undesirable, so the team changed the algorithm. The paragraphs that follow describe this change.

Let the term *member* of a distributed application mean a process that must coordinate its LoS with other members so the distributed application can function properly. Let each LoS on each member of the application correspond to the same level on any other member. Only if all members can provide the same LoS on all hosts will that LoS be provided by any of the hosts.

A useful tool in solving this problem is the concept of *invalidating* LoS with regard to another member. If one LoS cannot be provided to a member, then that level should be invalidated for all other members of the distributed application. When an LoS is able to be provided, the corresponding LoS can be validated for the other members. For each LoS, the system tracks whether the level is valid with all other members of its distributed application. For distributed applications with up to 32 members, this is done with a bit-field modification of an integer. If the user sets a bit to 1 to invalidate an LoS to another member, then the user can check the integer for a zero value, which means the LoS is valid to all other members.

The problem is solved by modifying the Maximize() algorithm as follows:

- All LoS for nondistributed processes are always valid. An LoS must be valid with all members of the distributed application to be moved to the active list in any step. All distributed applications begin with all LoS in the valid state.
- In step 2 of Maximize(), perform the following for each service that is selected to be moved to the active list and is a member of a distributed application:
 - Validate: If the level was previously invalidated with the member at hand, then send a message to LMs hosting all members of that distributed application. This will validate the corresponding LoS with the aforementioned member. Only move the service to the active list if it is valid with all members; otherwise, pass over this LoS and examine the next one in the inactive list.
- In step 3 of Maximize(), perform Validate each time a distributed application's LoS is selected to be upgraded,
- After step 3 of Maximize(), for LoS for distributed applications not placed in the active list in the present call to Maximize(), send messages to all members to invalidate corresponding LoS with regard to the member at hand.

Note

The above solution assumes that selected service can be provided on the local host.

6.2.2 Algorithm: Resource Measurement and Exponentially Moving Averages (EMA)

The LM measures resource availability and per-process use for CPU cycles in percent, memory in kilobytes, and network bandwidth in kilobytes per second. These measurements are taken periodically with a dynamically configurable interval. For resource availability and use for each service, three values are maintained for each resource:

- The current or last-measured value is recorded.
- The sum of all current values since the last calculation of an EMA is maintained for each resource. This value is used to calculate an average of all measurements taken since the last EMA calculation; it is possible for the period of EMA calculations to be greater than that of resource measurement.
- An exponentially moving average is maintained.

The formula used to calculate EMA follows:

$$\text{New EMA} = (1 - C) * \text{Old EMA} + C * \text{New Value}$$

Where:

- New Value = Sum of measured values since last EMA calculation/number of measurements
- C is a constant between 0 - 1 inclusive. The closer C is to 1, the more weight that is given to new values; conversely, the closer C is to 0, the more weight that is given to old values. After expanding a few iterations of the formula, old values exponentially decrease their weighting with the constant C.

The DSC monitors available resources and the consumption of resources by processes under its control. However, it cannot monitor resources outside its control; therefore, DCS sees these resources as decreased resource availability. Consequently, the sum of resources used by all processes that DSC monitors will never equal the difference between total and available resources. If a costly process is started outside of DSC's control, then DSC will see it as decreased resource availability. Although unaccounted for, decreased availability of resources will cause DSC to adjust LoS to adapt to the anomaly.

6.3 tcp_comms Application Programmers Interface and Users Guide

This document will help users write programs using `tcp_comms`. `tcp_comms` is a fast, transmission control protocol/Internet protocol (TCP/IP)-based message-passing library that uses a process-group model to help send and receive messages in a cooperative, distributed environment.

6.3.1 tcp_comms Functions

- **int tcp_comms_init(void)**
 - initialize `tcp_comms` structures and threads. Returns 0 on success; otherwise, -1. Will exit(-1) if it cannot get system information, such as hostname etc.).
- **int tcp_comms_join_group(char *name)**
 - Given a character string, registers the group specified by that string with the nameserver (NS) as a member of the group name. Blocks indefinitely until NS returns a gid. It is not thread-safe. It will break if > 1 thread simultaneously does a `tcp_comms_join_group()` and/or a `tcp_comms_get_group_id()`.
- **int tcp_comms_get_group_id(char *name)**
 - Ask NS for gid of group, "name". Blocks indefinitely until NS replies. Returns gid if group exists; otherwise -1 not thread-safe. It will break if > 1 thread simultaneously does a `tcp_comms_join_group()` and/or a `tcp_comms_get_group_id()`.
- **int tcp_comms_leave_group(int gid)**
 - Unregisters process with group corresponding to gid. Not implemented.
- **int tcp_comms_msg_send(int gid, void *msg, int n, int send_to_self_flag)**
 - Send message (of size n bytes) to group, gid. If `send_to_self_flag` is true, then also send this message to yourself if you are a member of gid. Returns 0 on success; otherwise -1.
- **int tcp_comms_get_next_msg(int gid, void **msg, int *n)**
 - Get next message from group gid and put address in *msg, size of msg in *n. It returns 0 on success. If there is no message for that gid, it does not block, but puts null in *msg, 0 in *n and returns -1. msg must be free()-ed when done.
- **int tcp_comms_wait_for_next_msg(int gid, void **msg, int *n)**
 - Get next message from group, gid, and put address in *msg, size of msg in *n. If there is no msg for the group, it blocks indefinitely. Always returns 0. msg must be free()-ed when done.
- **void tcp_comms_wait_forever(void)**
 - Simple routine that allows the program to finish the main thread without exiting the process. Usually as soon as it reaches the end of the main, it exits. This command stops this from happening.

6.3.2 How to Run tcp_comms

The following paragraphs describe the list of commands needed to be executed for `tcp_comms` to work with any program:

- 0) `setenv NS_SERVICE_PORT` to whatever you want it to be. This is the TCP port to which NS will listen. Defaults to 26660.
- 1) Start NS.
`cd $tcp_comms_home; ns &`

Where:

- `$tcp_comms_home` is an environment variable whose value is the path where `tcp_comms_home` resides.
- 2) `setenv NS_HOST` to the machine where NS is started. The user may have the equivalent of different groupname domains by starting multiple NS on different hosts and setting `NS_HOST` to whichever host the user's particular application wants to connect to or set `NS_SERVICE_PORT` to a different value. NS is not currently fault tolerant or redundant.
- 3) Run the application.

6.3.3 How to Link Programs with `tcp_comms`

The following paragraphs describe the actions needed to compile and link `tcp_comms` into a program that will be using it:

- `#include <tcp_comms.h>` in program source.
- For compiling, use a command similar to:
`cc -I$tcp_comms_home <your_file_name>.c ...`
- For linking, use a command similar to:
`cc -o <your_file_name> <your_file_name>.o
 — -L$tcp_comms_home -ltcp_comms -lsocket lnsl -lm
 — -posix4 -lthread`

Where:

- `$tcp_comms_home` is an environmental variable whose value is the path where the `tcp_comms` library, `libtcp_comms.a` resides, e.g.,
`$tcp_comms_home = /proj/dsc/tgeigel/tcp_comms)`

6.4 Demonstration User's Guide

The Advanced Technology Laboratories wrote this Demonstration Users Guide to help users run the DSC and to convert non-compliant applications to DSC-compliant ones. The guide has six sections:

- 6.4.1 **How to Run DSC** includes basic instructions on what processes to start and how and where to start them.
- 6.4.2 **Starting DSC Applications** describes the SM's GUI for starting up DSC-compliant applications.
- 6.4.3 **Local Manager (LM) GUI** explains the LM GUI and how to use it to control the DSC-compliant applications.
- 6.4.4 **Application Resource Files** describes the format of Application Resource files for DSC-compliant applications.
- 6.4.5 **DSC Application Programming Interface (API)** describes the DSC API functions to make a distributed application DSC-compliant.
- 6.4.6 **Final DSC Demonstration Setup Procedure** provides step-by-step instructions on how to run the final DSC demonstration

6.4.1 How to Run Distributed System Control

This example assumes users have three machines: machine A, machine B, and machine C. `$DSC_HOME` is an environment variable whose value is the path where the DSC software resides:

- `tcp_comms` must be started (Ref. Appendix 6.3).
- Execute the following commands on machine A:
 - `setenv NS_HOST machineA`
 - `setenv NS_SERVICE_PORT 26660 (optional)`
 - `setenv tcp_comms_home <path where tcp_comms resides>`
`$tcp_comms_home /ns &`
- Start SM on any machine on the network; for this example, SM will be started on machine A. Before running the SM, set the `NS_HOST` environment variable to tell `tcp_comms` on which host to find the NS:
 - `setenv NS_HOST machineA`
 - `setenv NS_SERVICE_PORT 26660 (optional)`
- Run the executable, `smd`:
 - `cd $DSC_HOME/SM`
 - `smd &`
- Start LMs on various hosts; in this example, machine B and machine C. You must login as root to run the LM.
- Before running each LM, set the `NS_HOST` environment variable to tell `tcp_comms` on which host to find the NS:
 - `setenv NS_HOST machine A`
 - `setenv NS_SERVICE_PORT 26660 (optional)`
- Run the LM with monitoring turned on (-M):
 - `cd $DSC_HOME/LM (see note 3)`
 - `./lmd -M &`

³Usually put the `lmd` in a directory local to the machine, such as `/local/machineB/DSC/LM`. Because the directory from which `lmd` is run is the default place to look for resource files, the team put them in the same directory.

- Type `lmd -help` to get help on running: `lmd -help`

```
lmd [ -M ] [ -m ] [ -e ema_period ] [ -r ServiceDir ] [ -p
PollPeriod ] [ -s StartDelay ] [ -n NetOption ] [ -l LogOption ]
```

Where:

- `-M` monitors statistics for individual processes; default is not to monitor individual processes.
- `-m` when LM terminates; do not remove processes started (messy terminate).
- `-r ServiceDir` defines the path for LM to search for application service files. If `-r` is not specified, LM checks the `APP_SERV_DIR` environment variable. If the environment is not set, LM looks in the current directory.
- `-p PollPeriod` is the period (in seconds) that LM polls applications for status
- `-s StartDelay` is a delay (in seconds) that LM waits before gathering network file descriptor data. Currently not used.
- `-n NetOption` is the net monitoring option for LM. Defaults are 10 seconds inactive, 1 sec active. The valid options are:
 - active: time (in seconds) net monitoring is active.
 - inactive: time (in seconds) net monitoring is inactive.
 - promisc: flag to turn on/off promiscuous mode: on=1, off=0
 - continuous: flag that overrides the previous netstat options. Network is monitored continuously,
 - off: do not monitor network.
- `-l LogOption` is the logging option for LM. The valid options are:
 - Console: write log message to console.
 - File = filename: write log to file filename.
 - Memory: write log to memory buffer. When terminated, LM writes memory buffer to `lm.log` file.
 - msg = logserver: write log to logserver group of `tcp_comms` group.

6.4.2 Starting DSC Applications

The SM is used to start processes. For each active LM, an entry will appear in the HOSTS list of the SM. Click on a host to bring up a dialog box (Figure 6-19) The dialog box lists all the DSC applications on that host and some performance statistics. A title bar on the dialog allows users to perform the following actions:

- **Start App** brings up a file selection box (Figure 6-20) which, when completed by the user, starts the entered application with arguments.
- **Stop App** kills the application selected in the process list.
- **Zap App** kills the application selected in the process list.
- **Zap All** kills all DSC applications on this host.
- **Restart All** kills and restarts all DSC applications on this host.

6.4.3 Local Manager Graphical User Interface

Figure 6-21 shows the LM display. The View menu button lets users bring down a Control Panel, which allows users to modify LM parameters with the Update button (Figure 6-22). See Appendix 6.2 for technical details on the modifiable parameters.

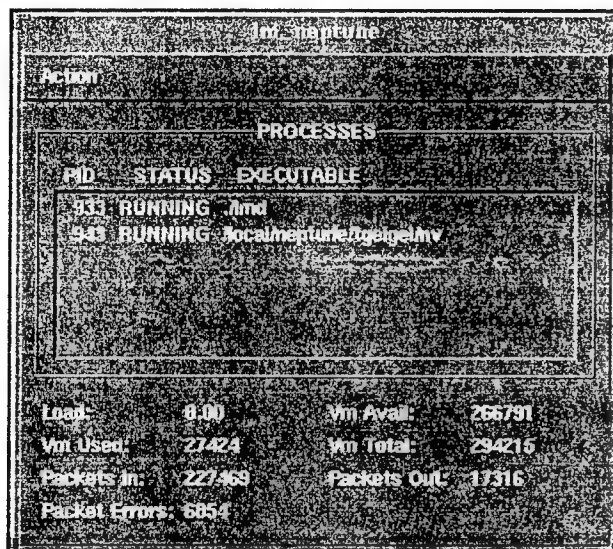


Figure 6-19. SM's LM window.

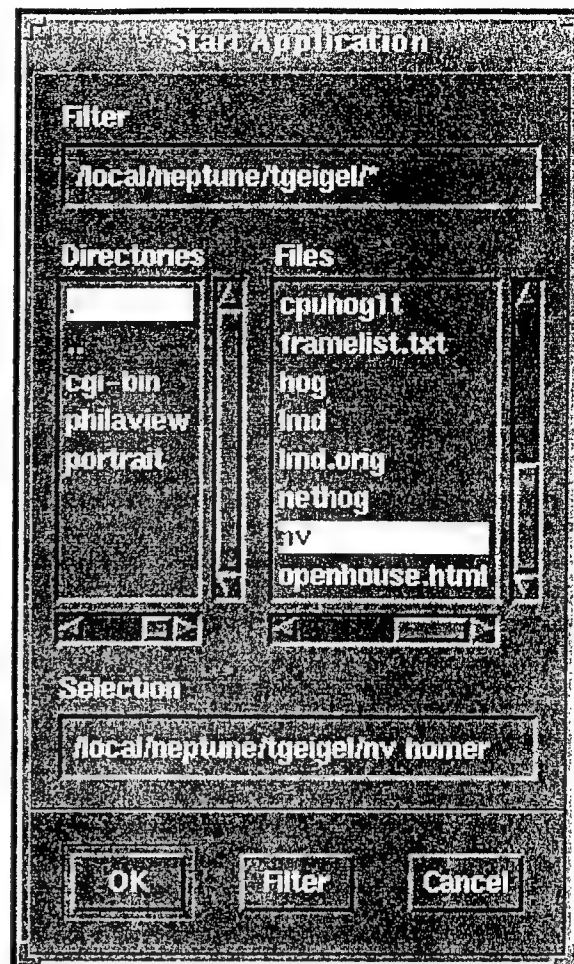


Figure 6-20. SM's Start Application window.

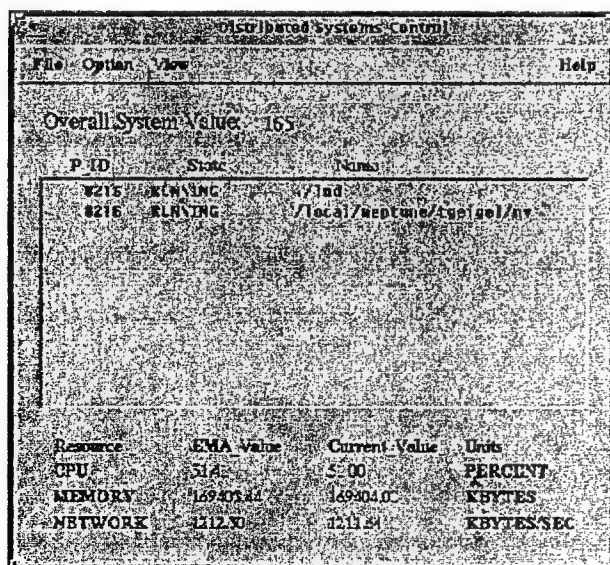


Figure 6-21. LM's GUI.

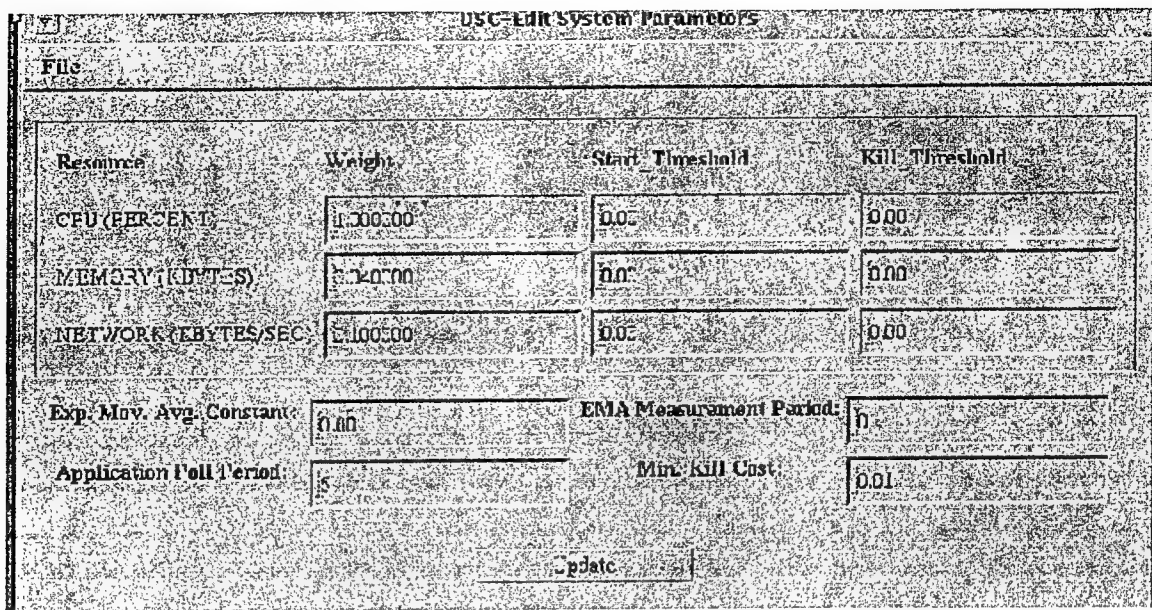


Figure 6-22. LM's Control Panel window.

Modifiable parameters are the following:

- **Weight** — These are the weighting factors by which each raw amount of resource is multiplied to get the weighted cost for each process. If these values are negative, then each resource is additionally multiplied by the degree to which that resource is being used by the system.
- **Start Thresholds** — Values by which available resources are reduced before running the Maximize algorithm. This enables a buffer amount of spare resources to prevent resource over use.
- **Kill Thresholds** — Used previously but is now inactive.
- **Exp. Mov. Avg. Constant** — This is the constant used to calculate EMAs for available system resources.
- **Application Poll Period** — Frequency with which measurements are taken for both host and per process resource figures.
- **EMA Measurement Period** — Frequency with which EMA calculations are performed. This is also the frequency with which the Maximize algorithm is called.
- **Min Kill Cost** — The minimum weighted cost a process must have to be suspended. Some processes have 0 weighted costs; for example, the LM. Do not suspend this parameter.
- **Update** — Updates the above values.

The main display has the host's overall value, a list of processes running under DSC control, and available resource statistics.

If the user double clicks on one of the applications in the list, an application dialog box will appear that lists the processes' LoS and associated values (Figure 6-23).

On the application dialog:

- The slider bar for priority is non-functional.
- A box with a slider bar denotes each LoS bar for value. Users can change the slider any time. The line on the top of the box denotes the following information:

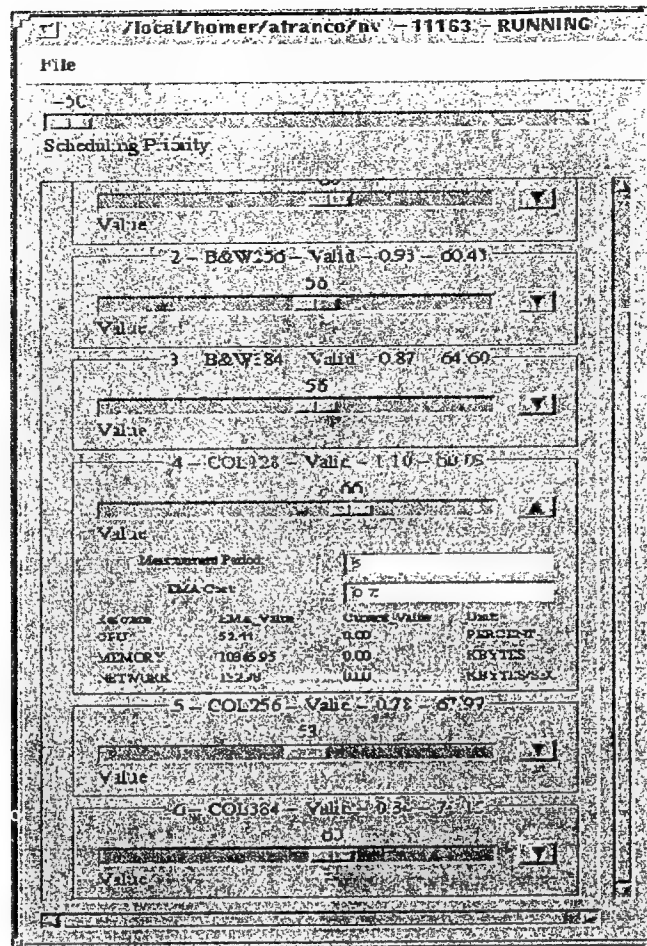


Figure 6-23. LM's Application GUI window.

— <Level id #> <Level name> <Valid/Not Valid> <Value/Cost Ratio> <Weighted Cost>

This line is highlighted in yellow for an active LoS.

- If the user clicks on the down arrow, the box will expand to display more information, such as the resource use statistics for this process. In this box, the user can also modify values for the EMA constant and measurement interval for this particular LoS. All of this information is stored in the resource file on termination of the process.
- The Option menu from the main window has three choices:
 - **Suspend:** Not implemented.
 - **Kill:** Kills the selected process in the process list.
 - **Resume:** Not implemented.

6.4.4 Application Resource Files

Each application has to have an associated resource file available to the LM. This file defines the application's LoS to the LM. The resource filename is expected to be:

.<application_name>_srv

For example, NV, the teleconferencing application, would have a corresponding `.nv_srv` file to define its LoS.

The format of the resource file is free-form ASCII text. The “#” character is a comment character. Any line starting with a “#” is ignored. Each LoS is defined as follows:

```
<LoS Name> <resource1 usage> <resource2 usage> <resource3 usage>
...
```

Where:

- `<LoS Name>` is a descriptive name for the LoS (It appears in the application’s dialog box.)
- `<resourcen usage>` is the estimated amount of resource used for the particular LoS, in absolute terms (e.g., percent of CPU use, actual memory use in bytes, actual number of packets used). For the initial version of DSC, `resource1` is CPU use, `resource2` is memory use, and `resource3` is network bandwidth.

A **CONSTANTS** line is used when defining the `ema_period` and the `ema_constant`.

- If a **CONSTANTS** line occurs before any LoS has been declared, then it defines the default values for these parameters for the entire application.
- If no **CONSTANTS** line occurs before an LoS is defined, then DSC default values are used.
- If a **CONSTANTS** line occurs after an LoS, then it defines the `ema_period` and `ema_constant` for that particular LoS. A **CONSTANTS:** line is in the form:

```
CONSTANTS: <period> <ema_const>
```

Where:

- **CONSTANTS:** is the keyword and must begin a **CONSTANTS** line
- `<period>` is the `ema_period` in seconds. This defines how often the system calculates measurements
- `<ema_const>` is the `ema_constant`. This value weights the importance of previous measurements.

See Appendix 6.2 for details on the EMA algorithm, including detailed information on the `ema_period` and `ema_constant`. Figure 6-24 shows a sample resource file.

6.4.5 DSC Application Programming Interface

To write DSC-compliant applications, users must select an application that can run under a variety of configurations, or LoS. For example, in the NV video-teleconferencing application, frame rate, color, and frame size were some of the configurable parameters that affected resource use. After defining the application’s LoS, users must insert the proper DSC functions into the application so that it can receive and handle LM messages. The DSC API provides functions for users to set up the application to asynchronously receive LM messages as part of a main control loop, as part of a polling strategy, or as part of an automatic callback strategy. The DSC functions are the following:

```

# This is an example application resource file for a DSC application.
# Default application specific values will be used unless a CONSTANTS
# line is inserted before LoSs description lines. A
# CONSTANTS line specifies values that are applicable to the overall
# application.
CONSTANTS: 100 .5
#         period ema_const
#
#
#   ema[i] = (1 - ema_const) * ema[i-1] + ema_const * x[i]
# There is one more line for each LoS which the
# application will provide. They will be formatted as follows:
# Name value resource1 resource2 resource3
level3 90 .95 9032446 1234567
level2 80 .65 7654321 912345
level1 70 .50 6543210 765432

```

Figure 6-24. Sample DSC application resource file.

- `int dsc_register(char *name, int (*service_cb)())` — Registers the application with the LM by setting up communications with LM via `tcp_comms` and creating an LM message handler thread. `name` is the name of the application and is used in naming `tcp_comms` process groups internal to DSC. `service_cb` is a callback function that the LM message-handler thread executes. It handles LM's `change_level` messages. Returns 1 on success, 0 otherwise.
- `int dsc_get_initial_service(int argc, char *argv[])` — Sets the application to LoS value if specified in application's argument list. The command line switch `-s <LoS>` specifies that the application initially run at the LoS represented by `<LoS>`, where `<LoS>` is an integer. Returns the actual LoS value of the application, regardless of whether it was successfully changed.
- `void dsc_configure_done()` — Sends a message to LM stating that the application is finished with its configuration. Not necessary to use, as it presently forces LM to only print out a log message.
- `void dsc_configure_done_and_wait()` — Like `dsc_configure_done()`, except that it blocks indefinitely. Used by multithreaded applications to stop the process from exiting.
- `void dscapp_cleanup(int signo)` — Called if LM sends the UNIX signal, `SIGHUP`, to the application, and the user wants the application to continue. Cleans up the LM log from the application. `signo` is unused.
- `void dscapp_bye(int signo)` — Called if LM sends the UNIX signal, `SIGKILL`, to the application, and the user wants the application to continue. Cleans up the LM log from the application. `signo` is unused.
- `int dsc_msg_handler(void)` — Checks for an incoming message from LM and processes it if there is one. Returns 1 if there is a message or 0 if no message is processed.
- `int dsc_wait_for_msg_handler(void)` — Like `dsc_msg_handler`, except this blocks indefinitely waiting for an incoming message from LM. Returns 1.

6.4.6 Final Demonstration Setup Procedure

The purpose of the final demonstration is to show that DSC can independently control multiple teleconferencing inputs. The demonstration consists of three machines, each

running an LM. Because of availability, only two machines have video cards and cameras installed. The third machine runs the SM.

The NV runs on all three machines in receive-only mode on the machine without the card, with the other two NVs sending their input to it instead of each other.

Setup consists of the following:

- For all machines, there should be a `/home/users/geigel/rome` directory, which is where all pertinent DSC software resides. The environment variable, `DSC_HOME`, is set to this directory. For each machine, there is a `<machine_name>_local` directory, where the operator will go to start the appropriate DSC processes.
- The following steps are needed to run the demo:
 - On the SM machine:
 - `cd $DSC_HOME/<SMmachine_name>_local`
 - `ns& (start tcp_comms' NS).`
 - `smd& (start up the SM).`
 - `lmd& (start up the LM for the SM machine).`

Note

lmd accesses local statistics from the kernel. Because of that, it is a setuid program owned by root. On occasion, the lmd will abort with a Segmentation Fault error. If this occurs, restart the NS and smd, then login as root and start the lmd as root)

- On machine A:
 - `cd $DSC_HOME/<machineA_Name>_local`
 - `lmd` start up the LM for machine A. See note above).
- On machine B:
 - `cd $DSC_HOME/<machineB_Name>_local`
 - `lmd& (start up the LM for machine B). See note above).`
- On SM machine:
 - Start up NV using the SM's GUI:
 - o On the SM GUI window labeled System Manager, double click on the lines `lm_<machine_name>` to bring up the SM Control Dialog for each machine (labeled `lm_<machine_name>`)
 - o On the `lm_<SMmachine_name>` window, click on the Action menu bar and click on the StartApp menu item. This will bring up the Start Application Window. Enter:
`/home/users/geigel/rome/<SMmachine_name>_local/nv`
`-recvOnly <SMmachine_name>`

An empty NV window will pop up. This is the receiving NV process running on SM machine.

- o On the `lm_<MachineA_name>` and `lm_<MachineB_name>` windows, do the same as `lm_<SMmachine_name>`, except enter the command:

```
/home/users/geigel/rome/<machine_name>_local/nv
<SMmachine_name>
```

The network video windows will appear on machine A and machine B with a video image from the cameras. The SM machine's NV will display two video images, one from each of the two machines.

- o On the `lm_<MachineB_name>` window, start up a `cpuhog` process by entering:

```
/home/users/geigel/rome/<MachineB_name>_local/cpuhog
50
```

A line for `cpuhog` will appear on machine B's LM GUI. The argument for `cpuhog` tells it for how many milliseconds to use the CPU. The LoS that the `cpuhog` runs under determines the duty cycle for CPU use:

```
--- #milliseconds of CPU use = argv[1]
--- #milliseconds of sleep = (LoS value * argv[1])/2
```

Initially, `cpuhog` runs at level 1, which means that its duty is 66 percent. If the LoS is changed to level 2, then the duty cycle is 50 percent, and so on.

- On machine B:
 - On the LM's GUI (labeled Distributed Systems Control), double click the line for the `cpuhog` process. An LM Application Control window (labeled the same as the process filename and its execution state) will appear. By varying the value of the `serv_a` LoS to 0 and increasing the `serv_b` LoS to >50, the user can reduce the amount of CPU consumed by the `cpuhog`, essentially simulating different CPU load levels on machine B. The result is that DSC will dynamically change the LoS for machine B's NV process, causing NV to adapt to varying CPU loads. On the SM machine's NV, the video image for machine B will vary to reflect the observed changes.

6.5 Benefit-Loss Scheduling

6.5.1. Objective

The objective of this task is to design and implement a thread scheduler on Solaris 2.3 that schedules realtime threads to minimize the total benefit loss. The threads dynamically present their worst-case computation time estimates, deadlines, and benefit-loss functions (BLFs) associated with their next computation segments to be executed.

6.5.2. Approach

The following paragraphs outline the basic design of the benefit-loss scheduler:

- An application process creates multiple threads that share the same address space with the parent process.
- One of the multiple threads is designated as a scheduler thread, and the rest of the threads become worker threads (WTs) that are scheduled by the scheduler thread.
- The WTs dynamically present their scheduling parameters (worst-case computation-time estimates, deadlines, BLFs, etc.) to the scheduler thread through a shared data structure called the scheduling parameter table (SPT). After they present themselves, they suspend themselves.
- The scheduler thread periodically checks the SPT, determines which WT to execute next, and informs the Solaris 2.3 Kernel Thread Scheduler of the WT's identification.
- The Solaris 2.3 Kernel Thread Scheduler dispatches the WT requested by the scheduler thread. Note that there are at most two threads seen by the Solaris Kernel Thread Scheduler in the Ready RealTime Thread Queue in the kernel.
- Whenever the deadline of any WT is missed, the benefit-loss incurred due to the deadline miss is accumulated. If the accumulated benefit-loss gets higher than a predetermined benefit-loss limit, the scheduler thread displays the results and exits.

6.5.3. Thread Model

The following paragraphs describe the design details of the worker and scheduler threads:

- **Worker thread:**
 - Each WT is "non-preemptible" and has its scheduling status, such as RUNNING, READY, and RECOVERING. The scheduler thread recognizes this status, but the Solaris Kernel Thread Scheduler may not.
 - The WT stores its computation-time estimate, deadline, and BLF (to generate benefit-loss upon deadline miss) into the SPT. Then it sets its status to READY and suspends itself.
 - When it is resumed (by the scheduler thread), it repeatedly reads the real-time clock until the amount of time equal to its computation time has passed as if it is doing its computation during that period. Initially, this step was implemented using a timer. It is currently being modified to use the realtime clock. After the period, the WT generates new computation-time estimates and new deadlines, stores them into the SPT, sets its status to READY, and then suspends itself.
 - Every WT is independent of the others.
 - The BLF $bl_{<i>i</i>}$ associated with each deadline imposed by the WT, T_i has

the following simple cliff form:

$bl\langle i \rangle = bl\langle max \rangle$ if T_i did not finish before its deadline, di .
 $bl\langle i \rangle = 0$ otherwise.

- **Scheduler thread** — When activated, it does the following:
 - Checks if there is a WT in READY status whose deadline is going to be missed. If there is, computes an actual execution time for that WT and checks again if the deadline is going to be really missed. If so, then it sets the status of the WT to RECOVERING. In the current implementation, the recovery time for the WT that missed a deadline and is in RECOVERING is set to the absolute deadline of the WT, which will be missed by the WT.
 - Checks if there is any WT in RECOVERING status that must be reincarnated at present or any WT in RECOVERING status of which the reincarnation time has already passed. If there is, change the status of the WT to READY and set a new deadline for the WT, which should be calculated based on the reincarnation time.
 - Get the WT ID, which is selected upon the current scheduling policy, set the status of the WT to RUNNING, and then resume the execution of the WT by calling the Solaris 2.3 Kernel Thread Scheduler to dispatch the selected WT.

6.5.4. Scheduling Policy

To test the effectiveness of the benefit-loss scheduler, the team compared three scheduling algorithms, two simple non-preemptive benefit-loss-based scheduling algorithms, and a known non-preemptive algorithm:

- **Highest Benefit-Loss-to-Laxity-Ratio First (HBLLRF)** — The WT whose laxity/benefit-loss is highest will get scheduled first. Laxity of WT = Absolute Deadline of WT - Current Time - Computation Time of WT.
- **Highest Benefit-Loss First (HBLF)** — The WT whose benefit-loss is highest will get scheduled first.
- **Least Laxity First (LLF)** — The WT whose Laxity is smallest will get scheduled first.

6.5.5. Implementation

The following paragraphs describe the implementation of the BLF scheduling design. Solaris 2.3 Thread Library is used to program multiple threads within a process. Each WT is created as a BOUND thread (Ref. section 6.5.8):

- `thr_create()`
- `thr_suspend()`
- `thr_continue()`
- `thr_yield()`
- `thr_self()`

Every thread is created as a thread of realtime class to avoid all system interventions during execution.

A random number generator is used to generate computation-time estimates, actual computation times, and deadlines. To achieve better randomness, the current time in micro-second units is used as a random number. In the current version, the computation-time estimate is regarded as the actual computation time. This will be modified so that the actual computation time is less than or equal to the computation-

time estimate. The difference between the computation-time estimate and the actual computation time will be a random number.

6.5.6. Result Analysis

Whenever any WT is going to miss its deadline, its benefit-loss value is added to the total benefit-loss. If the total benefit loss exceeds some predetermined benefit-loss limit, the scheduler thread displays the elapsed time, total benefit-loss incurred, benefit-loss per millisecond, and then exits.

The results depend on the various scheduling parameter settings. However, most of time, the HBLLRF performs most effectively.

The scheduling parameters were set as follows for the experiment:

- 10 msec \leq Computation Time \leq 200 msec
- Deadline = Computation Time * 4
- 00 \leq Benefit Loss Value \leq 1000
- Benefit Loss Limit = 10000

Table 6-VI lists the results of the experiment.

Table 6-VI. Results of the Benefit-Loss Experiment

Run#	<i>BL/msec</i>		
	HBLLRF	HBLF	LLF
Run 1	2.21	6.56	2.86
Run 2	1.84	4.87	2.30
Run 3	1.84	8.36	2.84
Run 4	3.00	5.85	2.10
Run 5	2.78	5.37	2.63
Average	2.334	6.202	2.546

6.5.7. How to Partition the Application

The partitioning decision intrinsically depends upon the application designer. However, it would be a reasonable approach to partition a program at each point of the blocking statement (e.g., receive statement). For example, a typical realtime task contains a loop that consists of a data-receiving part, data-processing part, and result-sending part. In this case, the loop body executed in each iteration can become one partition.

6.5.8 Solaris 2.3 Facilities for Scheduling Processes and Threads

The following paragraphs discuss the Solaris generic thread scheduler and how to use it to implement the BLF scheduler:

- **Process scheduler (Kernel thread scheduler)** — This schedules kernel threads onto processors. Each kernel thread is categorized into one of three classes, as shown in Table 6-VII. When a user process is created, one initial Light-Weight Process (LWP) kernel thread is allocated to the user process and

inherits the class type and priority of the parent user process. User processes are either time-sharing class or realtime class. Kernel threads in the system class are responsible for the kernel activities. No LWPs are associated with kernel threads in the system class. The class-specific priorities are converted to the global priorities based on which kernel thread scheduler schedules kernel threads.

- **Light-weight process (LWP)** — This may be viewed as a virtual CPU for user processes or user-level threads. The LWP is a kernel thread for execution of user processes or user-level threads.
- **Solaris thread library** is the programmer's interface for multi-threading and the user-level thread scheduler on LWPs performs the LWP interface.
- **User-Level Threads** — New user-level threads are created by calling Solaris' thread-library function from the user process or another user-level thread. There are two types of user-level threads:
 - **Unbound thread** — When a user process or a user-level thread creates unbound threads, the default number of LWPs are also created, but the unbound threads are not mapped to the LWPs. The unbound threads are scheduled by the user-level thread scheduler with respect to the other unbound threads in the same process. User-level thread-scheduling strategy for unbound threads is fixed-priority scheduling with no adjustments and no kernel involvement. The kernel thread scheduler has no effect on scheduling of the LWPs. The user can change the priority of each unbound thread in the same process.
 - **Bound thread** — Because unbound threads are only scheduled within a process, they are not scheduled with respect to threads outside the process. By binding a user-level thread permanently to an LWP, scheduling of the user-level thread becomes equivalent to the scheduling of the kernel thread. Each kernel thread supporting the bound thread can have a unique scheduling class priority. This priority is visible to the kernel thread scheduler with respect to all the other kernel threads in the system.

Table 6-VII. Classes of Kernel Threads

Class Type	Time-sharing Class	System Class	Realtime Class
Global priority of kernel thread	Lowest	Medium	Highest
Priority of kernel can thread in the same class	Kernel thread scheduler changes the priority of the kernel thread dynamically according to the behavior of the corresponding process or user-level thread	Fixed in the kernel code and never changed	Only superuser can change
Time slice of kernel thread	User can change. Administrators specify default time slice for kernel threads Kernel thread scheduler assigns time slices of different lengths to kernel threads according to the priority User has no control	Seem to have a default size of a time slice (not clearly mentioned) different time slices to kernel thread	Administrators specify default time slices Superuser can assign

Scheduling Policy:

- Among different priority kernel threads: fixed priority preemptive
- Among same priority kernel threads: round-robin

MISSION OF ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

- a. Conducts vigorous research, development and test programs in all applicable technologies;
- b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;
- c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;
- d. Promotes transfer of technology to the private sector;
- e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.